LOLA: LLM-Assisted Online Learning Algorithm for Content Experiments

Zikun Ye [*]
University of Washington

Hema Yoganarasimhan
University of Washington

Yufeng Zheng
University of Toronto

July 26, 2024

Abstract

In the rapidly evolving digital content landscape, media firms and news publishers require automated and efficient methods to enhance user engagement. This paper introduces the LLM-Assisted Online Learning Algorithm (LOLA), a novel framework that integrates Large Language Models (LLMs) with adaptive experimentation to optimize content delivery. Leveraging a large-scale dataset from Upworthy, which includes 17,681 headline A/B tests, we first investigate three pure-LLM approaches: prompt-based methods, embedding-based classification models, and fine-tuned open-source LLMs. We find that prompt-based approaches perform poorly, achieving no more than 65% accuracy in identifying the catchier headline. In contrast, both OpenAI-embedding-based classification models and fine-tuned Llama-3 with 8 billion parameters achieve an accuracy of around 82-84%. We then introduce LOLA, which combines the best pure-LLM approach with the Upper Confidence Bound algorithm to allocate traffic and maximize clicks adaptively. Our numerical experiments on Upworthy data show that LOLA outperforms the standard A/B test method (the current status quo at Upworthy), pure bandit algorithms, and pure-LLM approaches, particularly in scenarios with limited experimental traffic. Our approach is scalable and applicable to content experiments across various settings where firms seek to optimize user engagement, including digital advertising and social media recommendations.

Keywords: LLMs, Content Experiments, Bandits, News, Digital marketing.

[*] We would like to thank Yuting Zhu and the participants of the 2024 UW–UBC Conference, 2024 Informs Marketing Science, and 2024 China India Insights Conference for feedback and comments that have significantly improved the paper. Please address all correspondence to: zikunye@uw.edu and hemay@uw.edu.

1 Introduction

Digital content consumption has seen unprecedented growth, leading to a proliferation of content across various platforms. In today's real-time digital environment, media firms and news publishers need automated and efficient methods to determine which content generates high user engagement and platform growth. This includes identifying the most appealing articles, the most attractive headlines, and the catchiest cover images (Symonds, 2017; Coenen, 2019). Traditionally, media firms and publishers have relied on experimentation-based approaches to address this problem. Broadly speaking, there are two types of experimentation styles—(1) Standard A/B tests and (2) Online learning algorithms or bandits.[1]

A/B test is the most straightforward experimentation method, where a firm allocates a fixed portion of traffic to different treatment arms, assesses the results, and then goes with the best-performing arm. This is also known as the Explore and Commit strategy, or E&C, where the firm explores for a fixed period using A/B tests and then commits to one treatment based on the results of those tests. This approach is widely used in the publishing industry. For instance, The New York Times recently built a centralized internal A/B test platform, ABRA (A/B Reporting and Allocation architecture), allowing different teams to experiment with their content; see Yang (2024) for more details. Another example is Upworthy, which has implemented extensive A/B tests to choose headlines for news articles (Matias et al., 2021). The main advantage of this approach is that it is trustworthy—with a sufficiently large amount of traffic allocated to the A/B test, the resulting inference is unbiased and accurate. However, a major drawback of this approach is the wastage of traffic—by assigning equal traffic to all the treatment arms during the exploration phase (including the poorly performing ones), the firm incurs high regret or low profits. This is especially problematic in the news industry since news articles tend to have short lifetimes and become stale quickly (typically within a day or two). Therefore, if a firm wastes a lot of traffic to learn the best article/headline, the article itself might become irrelevant by the end of the A/B test.

The second approach, online learning algorithms, or bandits, are able to address some of these problems. These algorithms dynamically adjust traffic allocation to articles/headlines based on their features, past performance, and associated uncertainty. This method provides a more efficient way to optimize content delivery by continually learning and adapting. For instance, both Yahoo News and The New York Times have successfully implemented contextual bandit algorithms to identify content that maximizes user engagement (Li et al., 2010; Coenen, 2019). Online learning algorithms typically outperform A/B tests in terms of overall regret or reward. These algorithms reduce the extent of exploration over time by moving traffic away from poorly performing treatments dynamically. As a result, they tend to switch to the best-performing arms quickly and incur lower regret compared to standard E&C strategies. This has been established both theoretically (Lattimore and Szepesvári, 2020) as well empirically (Li et al., 2010).[2]

Nevertheless, online algorithms also have notable drawbacks. In standard content experiments, these algorithms suffer from a cold-start problem. That is, firms typically start these adaptive experiments assuming

[1] We use the terms adaptive experimentation, online learning algorithm, and bandits interchangeably throughout the paper.

[2] Similarly, in the context of digital advertising, Schwartz et al. (2017) use field experiments to demonstrate that adaptive algorithms outperform the standard balanced A/B test.

that all arms are equally likely to be the best-performing ones. This can lead to a situation where more-than-necessary traffic is allocated to sub-optimal arms, leading to higher regret. Additionally, due to their probabilistic nature, these algorithms can inadvertently converge on a sub-optimal arm, especially when the experimental traffic is small. These issues arise because online algorithms are designed for large traffic regimes, where the initialization of arms has little impact on the asymptotic regret. However, as discussed earlier, there is limited traffic available for experimentation in the media industry since content becomes stale quickly. Therefore, the negative effects of the "equally effective" initialization do not diminish sufficiently during the time horizon of interest, leading to higher regret.

Given these issues with content experimentation, the main question that this paper asks and answers is whether we can leverage the recently developed Large Language Models (LLMs) to enhance current content experimentation practices and simplify the process of identifying appealing content. LLMs, trained on extensive human-generated data, have demonstrated the ability to mimic human preferences and behavior in a variety of consumer research tasks (Li et al., 2024; Brand et al., 2023). Specifically, in the context of news articles, Yoganarasimhan and Yakovetskaya (2024) show that LLMs can correctly predict the polarization of news content. Hence, a relevant question is whether firms can directly use LLMs to predict the appeal of different content, potentially replacing traditional experimentation. This approach could significantly reduce the costs associated with experimentation while simplifying the content delivery processes.

To address this question, we leverage a large dataset of content experiments by the publishing firm Upworthy conducted in the 2013–2015 timeframe. The dataset consists of 17,618 headline tests spanning 77,245 headlines, 277 million impressions, and 3.7 million clicks. The goal of each A/B test was to identify the best headline among a set of candidate headlines (for a given article). This dataset is an excellent test-bed for our study for a few reasons – (1) it is based on an extremely large number of A/B tests over a high volume of impressions, (2) each A/B test received a relatively large number of impressions, allowing for accurate estimates of each headline's performance, and (3) it is sourced from a real media firm and represents the actions of a very large number of readers over a sufficiently long period to offer meaningful insights.

In the first part of the paper, we use this dataset to examine the extent to which pure-LLM-based approaches can accurately predict headline attractiveness. For this exercise, we focus on pairs of headlines (within a given A/B test) whose CTRs are significantly different (based on the observed data). This gives us a total of 39,158 pairs of headlines and allows us to recast the prediction task as a binary classification task, where a random guess gives 50% accuracy, and the perfect classifier reaches 100% accuracy. Thus, this serves as a preliminary testbed to evaluate whether pure-LLM-based approaches can predict which of the two headlines is catchier (or clickable).

We consider the three most widely used LLM-based approaches for this analysis—(1) Prompt-based approaches, (2) LLM text-embedding approaches, and (3) Fine-tuning approaches. For prompt-based methods, we consider two approaches: (a) Zero-shot prompting and (b) In-context learning. Zero-shot prompting is a technique in which an LLM generates responses without being explicitly trained on specific examples. This is the most common way in which most users interact with the LLM. In-context learning involves providing the LLM with a few demonstrations of similar tasks before generating responses for a specific task (Min et al., 2022; Xie et al., 2021). We find that GPT-3.5 performs as poorly as a random guess for both prompt-based approaches, with accuracy hovering around 50%. GPT-4 performs marginally better—it achieves an accuracy of 57.26% for zero-shot prompting and 60-65% for in-context learning. Overall, this suggests that prompt-based approaches cannot aid/replace content experiments in any meaningful way.

Next, we turn to the second approach: text embeddings. Specifically, we transform headlines into embedding vectors using the most recent embedding model from OpenAI. We then concatenate two embeddings as inputs, along with the winner with higher CTR as the output, to train binary classification models, including logistic regression and multilayer perceptron (a type of neural network). We find that OpenAI's embeddings combined with simple logistic regression can achieve around 82.50% accuracy, while more complex neural networks do not improve performance over the simple logistic model. Finally, we fine-tune a state-of-the-art open-source LLM — Llama-3 with 8 billion parameters, using low-rank adaptation (LoRA) (Hu et al., 2021). This approach performs the best, with 83.40% accuracy, which is a small improvement over the embedding-based approach. However, even the best-fine-tuned LLMs are unable to perfectly generate/predict which types of content are most appealing to users. As such, none of the LLM-based approaches can match the accuracy of experimentation-based approaches.

Therefore, in the second part of the paper, we propose and evaluate a novel framework for content experiments that combines the strengths of LLM-based approaches with the advantages of online learning algorithms, termed LOLA (**L**LM-Assisted **O**nline **L**earning **A**lgorithm). Our key insight is to use predictions from an LLM model as priors in online algorithms to optimize experimentation. The approach involves two steps. First, we train an LLM-based CTR prediction model using LoRA fine-tuning. The second step integrates these predictions into online learning algorithms. LOLA is a general framework, and depending on the specific goals of the experimentation, it can be adapted for different prediction models in the first step and different online algorithms in the second step. For example, if the goal is to minimize regret (i.e., maximize total reward/clicks), we propose a modified Upper Confidence Bound algorithm called LLM-Assisted 2-Upper Confidence Bounds (LLM-2UCBs) that is detailed in §4.2. The idea of LLM-2UCBs is intuitive: we can view LLM-based CTR predictions as auxiliary samples before the start of the online algorithm (e.g., a 1% CTR can be viewed as 100 impressions with 1 click or 1,000 impressions with 10 clicks). After fine-tuning this auxiliary impression size hyperparameter, we incorporate these auxiliary samples into the standard UCB algorithm (Auer et al., 2002) to obtain the reward estimator and shrink the upper confidence bound.

LOLA combines the advantages of both experimentation and LLMs, making it accurate and efficient. By integrating an LLM-based model to predict CTRs before the online deployment phase, we avoid wasting impressions on poorly performing headlines, especially in the initial stage when we face a cold-start problem. On the other hand, through experimentation, we can correct the errors in LLM predictions. This hybrid approach ensures that while LLMs provide reasonable initial predictions, ongoing experimentation refines and improves overall performance, making LOLA an effective solution for optimizing content delivery.

We present a comprehensive evaluation of LOLA and compare its performance with three natural benchmarks using the Upworthy dataset: (1) Explore and Commit, (2) a pure online learning algorithm, and (3) a pure LLM-based model. The first benchmark, E&C, was the status quo at Upworthy during the time of data collection, wherein the editors first ran an A/B test on a set of headlines for a fixed set of impressions and then displayed the winner for the rest of the traffic. For the pure online learning algorithm, we use the standard Upper Confidence Bound (UCB) algorithm, which is initialized with uniform CTRs. For the pure LLM-based approach, we use the LoRA fine-tuned Llama-3 to predict the CTRs of headlines and select the one with the highest CTR for all impressions, bypassing the experimentation phase. In LOLA, we use the same CTR prediction model as that used in the pure LLM approach and then employ LLM-2UCBs to maximize accumulated clicks. We find that LOLA outperforms all the baseline approaches, though the next-best algorithm varies based on the length of the time horizon (keeping the number of headlines fixed).

Specifically, when time horizons are small, both LOLA and the pure LLM-based approach beat the pure experimentation-based approaches (i.e., E&C and the UCB algorithm) due to relatively accurate LLM-based CTR predictions, and lack of sufficient time for the experimentation to provide meaningful updates. Specifically, both LOLA and the pure LLM outperform E&C by 8-9% and UCB by 6-7% in this scenario. However, as the number of impressions/time horizon grows, the performance of the pure LLM-based approach falls behind other experimentation-based methods because pure LLM-based approaches are static and do not take advantage of experimentation. In this case, LOLA beats pure LLM methods and E&C by 4–5%, and UCB by 2-3%. In sum, our experiments demonstrate that LOLA is able to leverage the power of LLMs early in the horizon and then build on it to take advantage of experimentation over time, thereby bringing together the strengths of both methods.

Finally, while we focus on regret minimization in a stochastic bandit setting given the Upworthy context, LOLA is a general-purpose framework and can easily accommodate different goals and/or settings. We provide variants of the standard LOLA (and results on the empirical performance of these variants when applicable) for a few natural extensions, e.g., for Bayesian bandit settings where the second stage algorithm can be Thompson Sampling instead of UCB, for the best arm identification problem where the goal is to identify the best content/headline rather than regret minimization, and for settings where the firm has access to user/contextual features that can be used to personalize content delivery.

In summary, our paper makes two key contributions to the literature. First, from a methodological perspective, we develop a novel framework, LOLA, to leverage LLMs for improving content experimentation in digital settings. To the best of our knowledge, this is the first paper that proposes combining LLMs with adaptive experimentation techniques. We demonstrate the value of our approach using a large-scale dataset from Upworthy when compared to baseline methods. Second, from a managerial perspective, LOLA can be used in a broad range of settings where firms need to decide which content to show. While we focus on the news/publishing industry given our empirical context, the LOLA framework is general and can easily extend to other settings, including digital advertising, email marketing, and website design. Further, the method is cost-effective, can be built on open-source LLMs (alleviating data privacy concerns), and is easily deployable across different domains once fine-tuned on relevant datasets.

Test ID	Headline	Impressions	Clicks
1	New York's Last Chance To Preserve Its Water Supply	2,675	15
1	How YOU Can Help New York Stay Un-Fracked In Under 5 Minutes	2,639	19
1	Why Yoko Ono Is The Only Thing Standing Between New York And Catastrophic Gas Fracking	2,734	34
2	If You Know Anyone Who Is Afraid Of Gay People, Here's A Cartoon That Will Ease Them Back To Reality	4,155	120
2	Hey Dude. If You Have An Older Brother, There's A Bigger Chance You're Gay	4,080	41

Table 1: Samples of Upworthy experiments' results for headlines. We display the key columns for our analysis, including the ID for the A/B tests, the text of headlines, and the number of impressions and clicks in the test.

2 Upworthy Data and Experiments

We now discuss the data sourced from Upworthy, a U.S. media publisher known for its extensive use of A/B tests in digital publishing. Upworthy conducted randomized experiments with each article published, exploring different combinations of headlines and images to determine which elements most effectively led to higher clicks. The archival record from January 24, 2013, to April 30, 2015, detailed in Matias et al. (2021), demonstrates how the packaging of headlines and images played a crucial role in Upworthy's growth strategy. Upworthy's editorial team created several versions of headlines and/or images for each article (internally called "package" by Upworthy). A package is defined as one treatment or arm for an article and consists of a headline, image, or a combination of both. Editors would first choose several of what they believed were the most promising packages to be tested. Then, they A/B tested these packages or treatments to identify which one resonated the best with their audience. During the A/B test, users only saw the headline (and an accompanying image in some cases), but not the article itself. Upon clicking on the headline, they were taken to the article. As such, the content of the article itself did not have any effect on users' clicking behavior. Upworthy's A/B test system recorded how many impressions and clicks each package received. Table 1 shows two examples of A/B tests with the relevant columns. The full details of all the A/B tests and their results are available from the Upworthy Research Archive.

We now describe the details of this dataset and discuss our pre-processing procedure. In the original dataset, there are 150,817 tested packages from 32,487 deployed A/B tests. The dataset records a total of 538,272,878 impressions and 8,182,674 clicks. All these statistics are also listed in the top panel of Table 2. These summary statistics indicate that during this period, each A/B test had an average of 4.64 packages, with each package receiving an average of 3,569 impressions and 54.26 clicks, resulting in an average click-through rate (CTR) of 1.52%. Within an A/B test, each package/treatment arm had the same probability of receiving an impression; so the number of impressions received by all the packages within a test is approximately the same. However, the actual number of impressions varies across tests, with the first quartile at 2,745, the median at 3,117, and the third quartile at 4,089. This is due to the relatively straightforward implementation of A/B tests, i.e., Upworthy did not conduct any power analysis in advance to determine the traffic for a given test, as confirmed by Matias et al. (2021).

Original data	# of tests	32,487
	# of packages	150,817
	# of impression in tests	538,272,878
	# of clicks in tests	8,182,674
Headline tests data	# of tests	17,681
	# of packages	77,245
	# of impression in tests	277,338,713
	# of clicks in tests	3,741,517
	# of headline pairs	140,655
	# of headline pairs with different CTRs at 0.05 significant level	39,158
	# of headline pairs with different CTRs at 0.01 significant level	23,682
	# of impressions (both test and non-test impressions)	2,351,171,402

Table 2: Basic summary statistics of Upworthy data during the time window between January 24, 2013 and April 30, 2015

Among these tests, some were conducted to test different images. However, the dataset does not allow us to trace back the actual images used in the tests. Therefore, we focus on the headline tests and filter out all the image tests. After this filtering, there are 17,681 tests, totaling 77,245 packages. On average, each article was tested with 4.37 headlines. Note that after filtering, each package or treatment simply refers to a unique headline; and therefore, we use the phrases package and headline interchangeably in the rest of the paper. The distribution of the number of packages/headlines for each article is shown in Table 3. After filtering, the dataset includes 277,338,713 impressions and 3,741,517 clicks; see the middle panel of Table 2. We also report the total number of impressions for the entire Upworthy website, as recorded in their data from Google Analytics. We see that the impressions attributable to the tests constitute only 22.9% of the total traffic.

# of headlines in one test	# of tests	% of samples
2	1,619	9.16
3	939	5.31
4	8,836	49.97
5	2,964	16.76
6	2,685	15.19
7 or more	638	3.61

Table 3: Distribution of number of headlines tested, across A/B tests.

To examine whether these A/B tests were informative, or not, we generate pairs of headlines and conduct pairwise t-tests on the CTRs of the headlines. For example, if four headlines were tested for one article, we generate $4 \times 3/2 = 6$ pairs of headlines, and test whether each pair is significantly different from each other;[3]

[3] The issue of multiple testing may arise when conducting a large number of pairwise comparisons, as it increases the likelihood of Type I errors, or false positives. In our dataset, tests involving 10 pairs or less (equivalent to no more than 5 packages) account

Test ID	Headline 1	Headline 2	CTR 1	CTR 2	p value	Winner
1	How YOU Can Help...	Why Yoko Ono Is...	0.0071	0.0124	0.054	2
1	Why Yoko Ono Is...	New York's Last...	0.0124	0.0056	0.009	1
1	How YOU Can Help...	New York's Last...	0.0071	0.0056	0.496	1
2	If You Know Anyone Who...	Hey Dude. If...	0.0289	0.010	0.000	1

Table 4: Example of pairwise comparisons for two A/B tests from the Upworthy data.

see Table 4 for examples of such tests for the two A/B tests highlighted earlier in Table 1. After generating pairs, we obtain 140,655 comparable headline pairs across all tests, of which 39,158 pairs significantly differ in CTRs at a 0.05 level and 23,682 at a 0.01 level. In summary, we find that only 28% of the tests were significant at the 0.05 level, and a mere 17% are significant at the 0.01 level. Overall, these results suggest that most pairs of headlines tend to perform similarly, and the task of learning which headline is catchier is non-trivial. To further illustrate the difficulty of this task, we conduct a survey, where we give human users pairs of headlines from the set of headline pairs that are significantly different from each other and ask them to identify the catchier headline. Even within this set of headline pairs, we find that the respondents' accuracy in the survey was not significantly better than random guessing. This further highlights the fact that the task of identifying engaging content is inherently challenging, even for humans. See Web Appendix §A for more details of the survey and its results.

3 Pure-LLM-Based Methods

In this section, we consider the 39,158 headline pairs that significantly differ from each other at the 0.05 significance level. We focus on these significant pairs for three reasons: First, insignificant pairs suggest that decisions in final headline selection do not substantially affect users' click behavior, implying that accuracy in predicting outcomes for "equally good" samples is not critically important from a business perspective. Second, the primary objective here is to evaluate various LLM-based classification methods to determine the most effective approach for the algorithm design in §4. As such, relative comparison across methods is more important as long as we use the same dataset for all methods. Third, as demonstrated in Web Appendix §D.2.1, our robustness checks confirm that including insignificant pairs in the training phase does not affect the overall performance of the methods considered. Furthermore, we note that all methods discussed in this section can be readily adapted from binary to multi-class classification tasks. Indeed, in §4 where we introduce our LOLA approach, we incorporate practical elements such as multiple headlines and insignificant pairs to ensure our experimental setup closely mirrors real-world conditions and that our conclusions are directly applicable to business scenarios.

In the rest of this section, we investigate the performance of three widely used LLM-based approaches—(1) GPT prompt-based approaches, including zero-shot prompting and in-context learning, (2) embedding-

for over 80% of all tests. The risk of multiple testing is relatively low when conducting fewer than 10 tests, and conventional significance levels, e.g., $\alpha = 0.05$ are generally adequate to account for this risk. Ultimately, our objective is to develop a more effective decision rule for assigning experimental traffic, and in §4, we utilize all available data without concern for multiple testing.

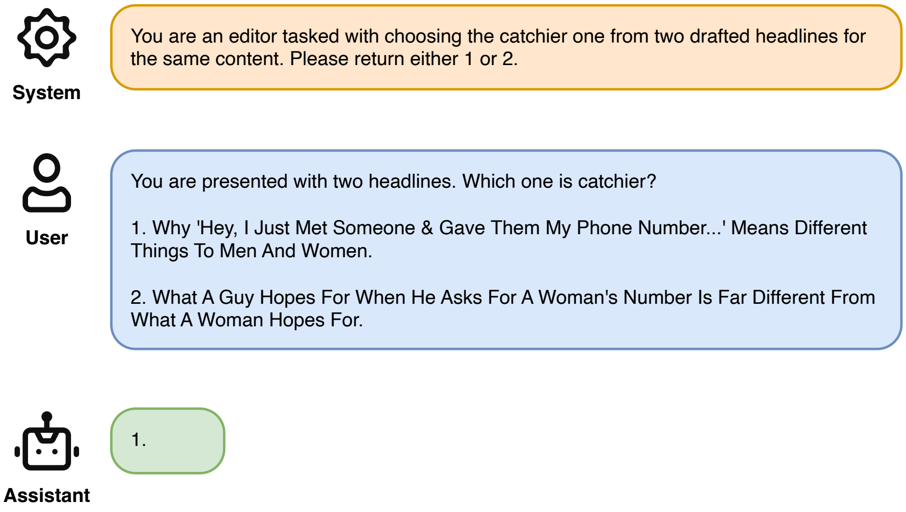
based classification models, and (3) fine-tuning of LLMs.

3.1 Prompt-based Approaches

Prompt-based approaches use GPT prompts to learn which headline is catchier, and there is no modeling or explicit fine-tuning involved. Prompt-based approaches have been the standard use-case for AI models in the marketing literature so far and have been used to examine if and when LLMs can simulate consumer behavior in market research studies (Gui and Toubia, 2023; Brand et al., 2023; Li et al., 2024). We consider two prompt-based approaches: (1) Zero-shot prompting and (2) In-context learning.

3.1.1 Zero-shot Prompting

Zero-shot prompting is a technique in which an LLM generates responses or performs tasks without being explicitly trained on specific examples. We evaluate the performance of headline pair comparison using zero-shot prompting based on OpenAI's GPT models. We do this through OpenAI's API in Python, which offers various parameters to customize and control the model's behavior and responses. The key parameters include specifying which GPT model to use (such as GPT-3 or GPT-4) and the system parameters, which guide the type of responses generated. The user's input or query is contained in the user parameter, while the assistant parameter holds the model's previous responses to maintain context in an ongoing conversation. Additionally, we set the default temperature parameter of 1.[4]



You are an editor tasked with choosing the catchier one from two drafted headlines for the same content. Please return either 1 or 2.

You are presented with two headlines. Which one is catchier?

1. Why 'Hey, I Just Met Someone & Gave Them My Phone Number...' Means Different Things To Men And Women.

2. What A Guy Hopes For When He Asks For A Woman's Number Is Far Different From What A Woman Hopes For.

1.

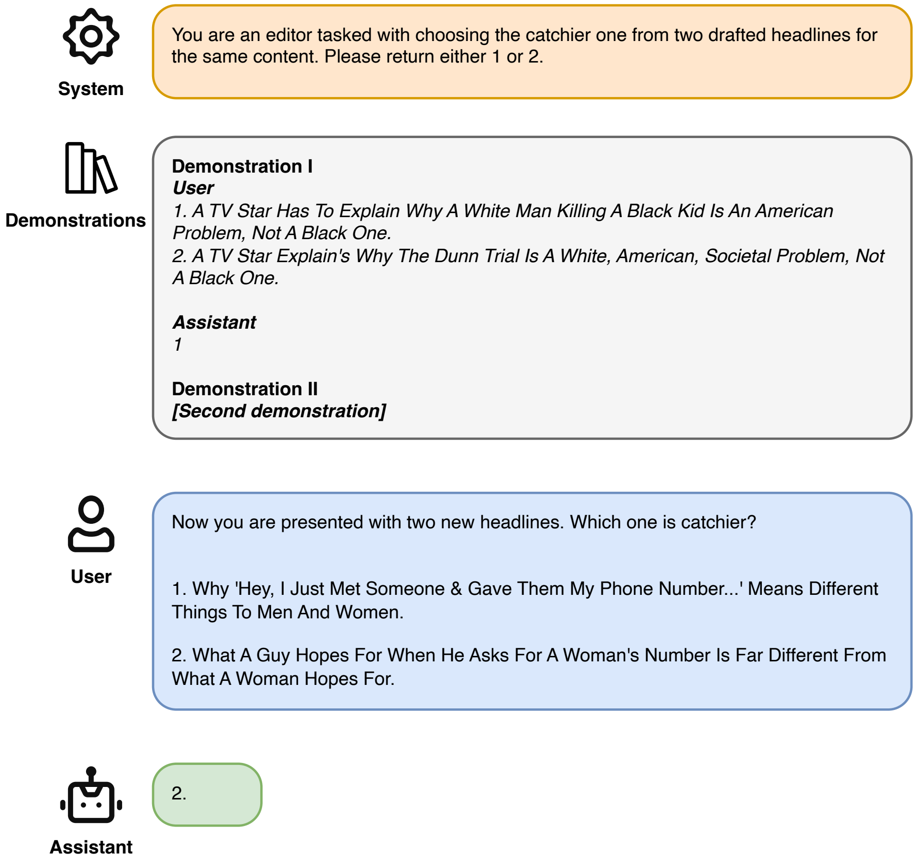
Figure 1: Zero-Shot Prompting for Headline Selection.

We design the zero-shot prompting to ask GPT which headline is catchier, as shown in Figure 1. We

[4] The temperature parameter scales and controls the randomness of the model's responses. A larger value results in a more diverse and creative output, while a lower temperature makes the output more deterministic and focused. For simplicity, we use the default setting with temperature as 1.0. We also tested a lower temperature of 0.1 to generate more deterministic outcomes, but we did not observe any significant difference in accuracy.

test this prompt using three OpenAI large language models,[5] each differing in parameter size: (1) GPT-3.5 (`GPT-3.5-turbo-0125`), which uses around 175 billion parameters, (2) GPT-4 (`GPT-4-turbo-2024-04-09`) which uses approximately 1.7 trillion parameters, and (3) GPT-4o (`GPT-4o-2024-05-13`), which uses an even larger number of parameters. The goal here is to investigate whether improvements in model architecture and parameter size can improve the accuracy of our predictions via standard prompts.

3.1.2 In-context Learning



You are an editor tasked with choosing the catchier one from two drafted headlines for the same content. Please return either 1 or 2.

Demonstration I
User
1. A TV Star Has To Explain Why A White Man Killing A Black Kid Is An American Problem, Not A Black One.
2. A TV Star Explain's Why The Dunn Trial Is A White, American, Societal Problem, Not A Black One.

Assistant
1

Demonstration II
[Second demonstration]

Now you are presented with two new headlines. Which one is catchier?

1. Why 'Hey, I Just Met Someone & Gave Them My Phone Number...' Means Different Things To Men And Women.

2. What A Guy Hopes For When He Asks For A Woman's Number Is Far Different From What A Woman Hopes For.

2.

Figure 2: In-Context Learning Prompt for Headline Selection.

In-context learning is a technique where a model is provided with demonstrations within the input prompt to help it perform a task. Demonstrations are sample inputs and outputs that illustrate the desired behavior or response. This method does not change the underlying parameters of the GPT models; rather, it leverages the model's ability to recognize patterns and apply them to new, similar tasks by conditioning on input-output examples. Previous studies have shown that in-context learning can significantly boost a model's accuracy

[5] As recommended by OpenAI, see this page, accessed on May 24, 2024, one can default to GPT-3.5-turbo, GPT-4-turbo, or GPT-4o. OpenAI notes that GPT-4-turbo and GPT-4o offer similar levels of intelligence.

and relevance in responses compared to zero-shot prompting. Xie and Min (2022) explains this using a Bayesian inference framework, suggesting that in-context learning involves inferring latent concepts from pre-training data, utilizing all components of the prompt (inputs, outputs, formatting) to locate these concepts, even when training examples have random outputs. Xie et al. (2021) further elucidate that in-context learning can emerge from models pre-trained on documents with long-range coherence, requiring the inference of document-level concepts to generate coherent text. They provide empirical evidence and theoretical proofs showing that both Transformers and LSTMs can exhibit in-context learning on synthetic datasets.

We design in-context learning prompts as shown in Figure 2. Our prompts include demonstrations of catchier headlines and ask the model to determine which headline is catchier for a subsequent pair of headlines. We use the headline with the higher CTR from the test, as indicated in the "Winner" column of Table 4, as the correct answer in our demonstrations. A few demonstrations are typically sufficient to guide the model. Improvement tends to have diminishing returns quickly, i.e., adding more demonstrations does not significantly enhance performance after a certain point (Min et al., 2022). In our numerical experiments, we use two or five demonstrations to investigate the impact of the number of examples on performance.[6]

An interesting aspect of in-context learning is the model's robustness to incorrect labels in the demonstrations. Min et al. (2022) show that randomized labels in demonstrations barely affect performance and highlight other critical aspects like the label space, input text distribution, and sequence format. This is likely because the model relies more on the distribution of input text, label space, and format rather than specific input-label mappings. That is, in-context learning seems to benefit from recognizing general patterns and structures in the data, which the model infers from its pre-training rather than the actual labels. To examine if this is the case in our setting as well, we also consider experiments where we intentionally flip the labels in the demonstrations provided to the model.

3.1.3 Performance Analysis of Prompt-based Approaches

Before presenting the results, we make note of two points. First, we do not find any significant difference between GPT-4 and GPT-4o; therefore, we only report the results from GPT-3.5 and GPT-4 for simplicity. Second, the GPT models can produce different answers even when given the exact same prompt. To mitigate this variability, we randomly select 1,000 significant pairs of headlines and conduct all our experiments on this same test set. Similarly, all the demonstrations are also randomly sampled and fixed for all experiments to reduce the variance of prompt outputs.

The results of all the prompt-based experiments are shown in Table 5. A more detailed pairwise comparison and t-test analysis of all the different experiments are available in Web Appendix §B.1. The second column of Table 5, n_{demo}, represents the number of demonstrations used for in-context learning, with $n_{demo} = 0$ indicating a zero-shot prompting. Additionally, for all the in-context learning experiments, we consider two scenarios—one where the demonstrations are shown with ground truth labels and another where they are shown with flipped outcome labels. The third column, "Flipped Label" indicates whether the labels

[6] We did run experiments with a larger number of demonstrations, but did not find any significant improvements from doing so; and hence do not report them here.

GPT Model	n_{demo}	Flipped Label	Accuracy	[0.025	0.975]
GPT-3.5	0	0	54.20%	51.10%	57.10%
GPT-3.5	2	0	48.70%	45.50%	51.50%
GPT-3.5	2	1	50.85%	47.60%	53.70%
GPT-3.5	5	0	51.30%	48.20%	54.30%
GPT-3.5	5	1	50.10%	46.80%	53.20%
GPT-4	0	0	57.26%	54.20%	60.00%
GPT-4	2	0	63.60%	60.50%	66.40%
GPT-4	2	1	62.90%	59.70%	65.70%
GPT-4	5	0	60.80%	57.70%	63.40%
GPT-4	5	1	61.80%	58.50%	64.80%

Table 5: Results for prompt-based approaches with different GPT models, the number of demonstrations, and whether the demo's label is flipped or not. Accuracy is defined as the ratio of correct answers divided by the number of prompts (1000 in our analysis). The 95% confidence interval is calculated by performing 5,000 bootstrap resamples of the entire 1,000 pairs. To save on the monetary cost of prompts and not introduce additional variance, we did not rerun the GPT prompts for the same samples; instead, we used the responses/predictions from the original run.

were flipped or not.

We find that prompt-based approaches using GPT-3.5, except for the zero-shot prompting, are not significantly better than random predictions (accuracy rates hover around 50%). Neither the number of demonstrations used nor the label type have any significant impact on the accuracy of the GPT-3.5 predictions. Interestingly, zero-shot prompting using GPT-3.5 is slightly better than random prediction and also marginally outperforms in-context learning. In comparison, experiments with GPT-4 have accuracy rates around 60-65%, which is a significant improvement in performance compared to GPT-3.5. For GPT-4, in-context learning significantly improves performance over zero-shot prompting in GPT-4, though increasing the number of demonstrations from two to five does not result in a significant difference in performance. There is also no significant difference between using correct labels and incorrect labels in the demonstrations for in-context learning. This confirms that in-context learning helps GPT understand the task better through the input-output examples, but it fails to leverage the information in mapping from inputs to outputs effectively.

In summary, we find that prompt-based methods have low accuracy (no more than 65%), high randomness (different answer outputs even under the same prompt inputs), slow running time (compared to other methods introduced later in this section), and high monetary cost (see Web Appendix §E for a detailed discussion of costs and running time). All of these drawbacks make them unsuitable for real-time business applications, i.e., these methods cannot replace experimentation for content selection in digital platforms.

3.2 Classification Models with LLM-based Text Embeddings

We now consider another LLM-based method for the prediction task. Specifically, we leverage the text embeddings from OpenAI and utilize classification models, including Logistic Regression (Logit) and Multilayer Perceptron (MLP), to predict which of the headlines in a pair is likely to generate a higher CTR.

Text embedding techniques transform large chunks of text, such as sentences, paragraphs, or documents, into numerical vectors. These embedding vectors can then be used in a variety of applications, including text classification, where texts are grouped into categories; semantic search, which improves search accuracy by understanding the meaning behind the search queries; and sentiment analysis, which is used to determine the emotional tone of a piece of text.[7] See Patil et al. (2023) for a more detailed survey of the most recent text embedding models and their applications.

An interesting question is why text embeddings obtained from LLM models trained for the next word/token prediction task with the cross-entropy loss are able to produce informative features for downstream classification tasks. Saunshi et al. (2020) provide an explanation by demonstrating that classification tasks can be reformulated as sentence completion tasks, which can be further solved linearly using the conditional distribution over words following an input text. Thus, the embeddings, which can be roughly viewed as the next token probability generated from a next-word prediction task, inherently capture the contextual information and relationships between words, making them useful for various downstream tasks such as sentiment analysis and headline selection (as our experiments illustrate).

3.2.1 Implementation Details

Figure 3 illustrates the conceptual framework of our approach here. We concatenate the embeddings of two headline embeddings[8] and apply the downstream classification models to predict the probability that the first headline will generate a higher CTR. We consider three embedding models:

- **OpenAI embeddings:** At the time of this study, the latest and best embedding model available from OpenAI was `text-embedding-3-large`. These embeddings are obtained using the OpenAI embedding API in Python. This embedding model generates embeddings with up to 3,072 dimensions and supports a maximum token limit of 8,191, with its knowledge base extending up to September 2021. A notable feature of this model is its flexibility – it allows users to tailor the embedding dimension to their specific needs.[9] In our experiments, we consider both 3,072- and 256-dimensional embeddings to examine whether the size of the embedding vector size affects the performance of the downstream classification models.[10]
- **Llama-3-8b:** Meta's Llama-3 (Meta, 2024b) is regarded as one of the most advanced open-source LLMs

[7] Embedding models and standard prompt-based LLMs used share some common features, e.g., the underlying Transformer architectures and extensive pretraining on large text corpora. However, they are optimized for different tasks. Models like GPT are optimized for generating coherent and contextually relevant text based on input prompts while embedding models focus on producing high-quality embeddings for downstream tasks.

[8] Because embeddings approximate the next token probability, one can also use the difference (subtraction) of two embedding vectors instead of the concatenation as the input to the downstream classification model. In our task, these two input types have no significantly different impact on the final classification accuracy.

[9] Other models capable of producing text embeddings, like BERT (Devlin et al., 2018) and Sentence-BERT (Reimers and Gurevych, 2019), have fixed embedding vector dimensions that remain constant once the model is trained. These dimensions cannot be changed or resized for different applications. However, OpenAI's embedding model uses a technique that, while producing outputs of fixed length, encodes information at different granularities (Kusupati et al., 2022). This allows a single embedding to adapt to the computational constraints of downstream tasks. More specifically, downstream tasks can use only the first several dimensions of the original vector, and the information retained decreases smoothly as a shorter part of the vector is used.

[10] A higher dimension does not necessarily mean better performance. While a higher dimension can retain more information for each sample, it also means that the samples are spread more sparsely in a higher-dimensional space given a limited number of training samples. This sparsity can increase the possibility of over-fitting.

to date (Meta, 2024a). We obtain embeddings from Llama-3 with 8 billion parameters (Llama-3-8b). We fine-tune this exact model using LoRA in §3.3; therefore, using the embeddings from this model allows for a direct comparison of fine-tuning vs. embedding-based classification methods (see §3.4 for comparisons). Llama-3-8b does not produce embeddings explicitly designed to represent the semantic meaning of entire input sequences. To address this issue, we apply the technique described in BehnamGhader et al. (2024), which transforms the Llama-3-8b model into a more suitable text embedding model, outputting embeddings with a dimension of 4,096.

- **Word2Vec embeddings:** Word2Vec, proposed by Mikolov et al. (2013), was the leading embedding model before the widespread adoption of Transformers in the Natural Language Processing (NLP) literature. Unlike LLM-based text embeddings, Word2Vec does not use Transformers and cannot pay attention to surrounding token information. Embeddings from Word2Vec have been used in the recent marketing literature in a variety of use cases, including product attribute modeling (Timoshenko and Hauser, 2019; Wang et al., 2022), the effect of creativity on product popularity (Sozuer Zorlu et al., 2024). We compute the Word2Vec embedding vector for each word in the headline and average them to get the headline embedding, a common practice in Word2Vec. To ensure consistency with the dimensions used for OpenAI embeddings, we consider Word2Vec embeddings with 256 and 3,072 dimensions.

Next, to assess the impact of model complexity on performance, we consider two classification models:

- **Logit model with linear features:** This is one of the simplest binary classification models and predicts the probability that a given input belongs to a particular class by applying the logistic function to a linear combination of the input variables.
- **Multilayer perceptron:** MLP is a powerful neural network architecture consisting of multiple linear and non-linear hidden layers that allow the model to learn complex patterns in the data. It employs a fully connected neural network with ReLU activation for hidden layers and Sigmoid transformation for the output probability, with dropout layers added to prevent over-fitting. We refer interested readers to Chapter 5 in Zhang et al. (2021) for more details. We train the MLP using the stochastic gradient descent algorithm to minimize binary cross-entropy and perform hyper-parameter tuning with Bayesian Optimization to determine the optimal configuration. Please see Web Appendix §C.1 for additional details of the fine-tuning process and the architectures used in the final implementation.

Finally, in terms of the data, as discussed earlier, we choose all headline pairs that are significant at the 0.05 level. Note that the same headline can appear in multiple pairs when there are more than two headlines tested in one experiment. To avoid information leakage, we first randomly split the data into 80% training and 20% test sets based on the test ID (the experiment ID for Upworthy) to ensure all headline pairs under the same experiment are either in the training set or the test set. Notice that Upworthy allowed additional experiments for the same headline under a new experiment ID because the editor may require an extra test to make an informed decision. It means we may have the same headlines in both the training and test sets, even if they were in different experiments. In such cases, we discard headlines that have appeared in the training

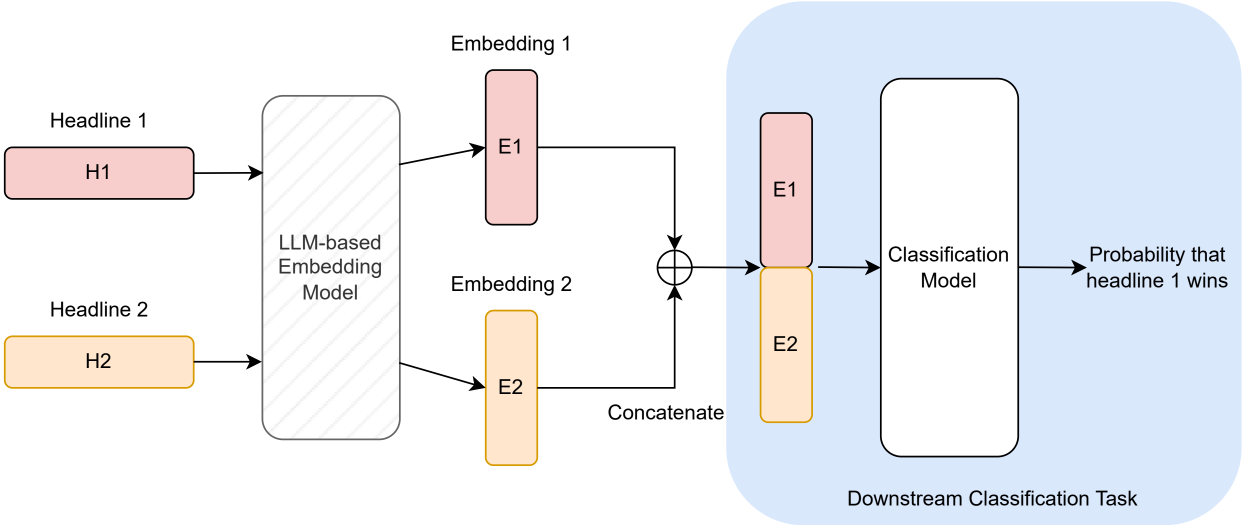

Headline 1
H1
Headline 2
H2
LLM-based Embedding Model
Embedding 1
E1
Embedding 2
E2
Concatenate
E1
E2
Classification Model
Probability that headline 1 wins
Downstream Classification Task

Figure 3: The pipeline of the classification with the LLM text embeddings. Headline 1 and headline 2 are natural language sentences while Embedding 1 and Embedding 2 are numerical vectors.

set from the test set to avoid information leakage.[11]

3.2.2 Performance Analysis of Embedding-based Approaches

In Table 6, we present the performance of various embedding-based classification models.

Embedding Model	Classification Model	Accuracy
OpenAI-256E	Logit MLP	77.31% 77.62%
OpenAI-3072E	Logit MLP	**82.50%** 82.26%
Llama-3-8b-4096E	Logit MLP	74.03% 79.40%
Word2Vec-256E	Logit MLP	67.89% 67.32%
Word2Vec-3072E	Logit MLP	67.55% 67.02%

Table 6: The performance of different embedding and classification models on test data. All models use the same training data, and accuracy is evaluated on the test data. Note that the Logit model's training is a deterministic process after fixing the data, while MLP training incurs randomness via stochastic gradient descents. Therefore, we rerun MLP training three times with different sample paths (training set shuffling) and report the average accuracy from three reruns.

[11] There are alternative ways to do the train-test split without discarding any samples. One approach is assigning one headline pair at one time to a training or test set and then adding all pairs with the same headline in the same group until all samples are allocated. However, after conducting several experiments in our study, we found that the splitting procedure has minimal impact on the final results. Hence, to maintain clarity and simplicity in our exposition, we simply discard around 25% of test headlines that have appeared in the training set.

At a high level, we see that all the embedding-based classification models perform better than prompt-based approaches. This performance can be attributed to the fact that these models are trained on a customized dataset, allowing them to learn decision rules tailored to the headline selection task. Because these models can leverage a larger labeled dataset during training, they are more generalizable and robust. In contrast, in-context learning relies on the LLM's ability to understand and generalize from only a few demonstrations. Furthermore, the task of choosing the catchier headline from two options may not be a common scenario in OpenAI's training corpus. Consequently, GPT models may struggle to generalize effectively from the limited examples provided in prompts, leading to less accurate predictions.

Next, we compare performance under different embedding models. We see that OpenAI embeddings outperform both Llama-3-8b and Word2Vec-based models. This is understandable since OpenAI's LLMs are state-of-the-art models that naturally perform better than the classic Word2Vec embeddings. Also, the performance of embeddings from open-source Llama-3-8b falls between that of the proprietary OpenAI model and the older Word2Vec that does not use the transformer architecture. We also see that for OpenAI, higher-dimensional embeddings give higher-quality encoding and, thus, higher prediction accuracy. However, there is some limit beyond which there are diminishing returns. In our experiment, the 3072-dimensional embedding (OpenAI-3072E) improves over the 256-dimensional embedding (OpenAI-256E) by around 5%. A more striking observation is that classification model complexity does not improve the prediction for OpenAI embeddings. Note that there is no significant difference in accuracy between MLP and Logit for the following embedding models – OpenAI-256E, OpenAI-3072E, Word2Vec-256E, and Word2Vec-3072E. But for Llama-3-8b embeddings, the MLP classification model outperforms Logit classification model by approximately 5%. We hypothesize that with more dimensions in text embeddings, the separation between different classes becomes more complex (e.g., higher sparsity and higher risk of over-fitting), requiring a more sophisticated model for classification. Overall, our findings suggest that a Logit classifier with linear features on top of fixed text embeddings from OpenAI can effectively capture the attractiveness of content.

We now investigate a few other aspects of these models and summarize our findings below. For brevity, we simply discuss the findings here and refer interested readers to the Web Appendix for details. Note that in all these analyses, we focus on OpenAI embeddings (since it offers the best performance) and the logistic regression with linear features as our classification model (since the MLP does not offer any significant improvement over the logit model for OpenAI embeddings and is easier to train).

- First, we examine the conditions under which these approaches fail. We use cosine similarity measures to quantify the similarity between the two headlines in a pair and find that these approaches tend to perform worse when the headlines are similar. This is understandable since text embeddings may not be able to capture the differences between two headlines when they are similar. As a result, it is hard for a classification model to differentiate between them. See Web Appendix §C.2 for details.
- Second, we quantify the role of the number of training samples on predictive accuracy. For the 256E-dimensional embedding, we find that approximately 65% of the training set, comprising approximately 25,000 pairs (with the remaining 35% used for the test set), is adequate to achieve high and stable

performance. However, when we use the 3072E-dimensional embedding, there is evidence of overfitting even when we use 95% of the data for training. This suggests that for larger embedding vectors, we need much larger training datasets. See Web Appendix §C.3 for details.

- Third, we examine whether we can improve the performance and alleviate any overfitting issues by adopting dimensionality reduction techniques (PCA) and/or regularization (adding L^1/L^2 norms). However, we don't see any significant improvement with these methods. See Web Appendix §C.4 for details.
- Finally, there are two potential threats to the inference from our analysis. First, one may be concerned as to whether Upworthy's data has been used as part of OpenAI's training corpus, which could potentially invalidate the results. A second concern is that readers' preferences changed over time and that the current sample splitting method fails to take into account when the headline was created. We examine both these issues and provide a detailed discussion and robustness checks to alleviate concerns around them in Web Appendix §C.5.

3.3 Fine-Tuning Open-Source LLMs with LoRA

In §3.2, we trained classification models using text embeddings as fixed inputs, bypassing the need to train the LLM itself. The best-performing embedding-based model reaches an accuracy of 82.50% without modifying any parameters in the LLM. This raises a natural question: if we can train LLMs directly, can we achieve better performance? However, training LLMs from scratch is costly and resource-intensive. Therefore, in this section, we turn to fine-tuning techniques, which involve partially adjusting the parameters in the LLM to improve performance while keeping the computational cost manageable. Note that to fine-tune LLMs, we need to be able to access the underlying model parameters. This is only feasible for open-source LLMs, like BERT and Llama.[12] The goal of fine-tuning is to adapt a general-purpose LLM for specific downstream tasks through a process typically involving supervised learning using labeled datasets.

For the fine-tuning task, we use Meta's Llama-3-8b as our primary base model. Zhao et al. (2024) consider a large-scale experiment where they fine-tune different open-source LLMs and show that for NLP tasks, including news headline generation, Llama-3 offers one of the best performance. Llama-3 offers two versions: one with 8 billion parameters (Llama-3-8b) and another with 70 billion parameters (Llama-3-70b). We opted for the smaller version since it is easier to fine-tune and, as we will see, even with this smaller model, we see evidence of over-fitting (see §3.3.3 for details). Hence, we avoid fine-tuning the 70-billion-parameter model, where over-fitting problems are likely to be exacerbated.

The rest of this section is organized as follows. We present the LoRA method for efficient fine-tuning in

[12] Some proprietary LLMs offer black-box fine-tuning. For instance, OpenAI allows fine-tuning GPT-3.5, GPT-4/GPT-4o (with limited experimental access) through its API in a black-box manner. Users provide the training and test datasets and select the base GPT model, but are not provided any details of the fine-tuning process or the underlying model parameter. This approach also has a number of other drawbacks – the fine-tuning API is quite expensive compared to the open-source fine-tuning approaches, the fine-tuned model cannot be hosted on a local machine, data privacy concerns since the media firm's proprietary data will need to be submitted to OpenAI. Given these drawbacks, it is preferable for firms to fine-tune open-source LLMs, as the process and data are fully under the control of the firm. However, for the completeness of comparison, we fine-tuned the ChatGPT-3.5 model on the training set using this API, with the default configuration provided by OpenAI, and the same training-test dataset split. We obtained an accuracy of only 79.80% on the test set.

§3.3.1. The goal is not to delve into every technical detail but to provide the necessary background information for a high-level understanding of fine-tuning. Then we detail our implementation in §3.3.2 and report our results in §3.3.3. Background information related to Transformer block, the fundamental component of most LLMs, including Llama-3, and the basic architecture of Llama-3 are given in Web Appendix §D.1.

3.3.1 LoRA Fine-Tuning Technique

Traditional fine-tuning of LLMs involves updating a substantial number of parameters, which can be computationally expensive and memory-intensive. To efficiently fine-tune the Llama-3 model on hardware with limited GPU memory, we employ Low-Rank Adaptation (LoRA), a popular Parameter-Efficient Fine-Tuning (PEFT) method developed by Hu et al. (2021), enabling practical fine-tuning of large-scale LLMs on medium-scale hardware.[13] LoRA operates under the assumption that updates during model adaptation exhibit an intrinsic low-rank property and introduce a set of low-rank trainable matrices into each layer of the Transformer model. This approach leverages the low-rank decomposition, which significantly reduces both memory usage and computational time through a significantly reduced number of trainable parameters.

We illustrate LoRA in Figure 4. We use W to represent any pre-trained weight matrix, which can be either W_Q, W_V, or W_K. Specifically, W_Q is the weight matrix for the query vectors, W_K is for the key vectors, and W_V is for the value vectors in the attention mechanism of the Transformer architecture. These matrices store most of the information and knowledge learned by LLM, and are crucial for calculating the attention scores and subsequently determining the relevance of different tokens in the input sequence. For more technical background, we refer to Web Appendix §D.1. We use $d \times d_W$ to represent the dimension of the weight matrix and r to represent the rank of the low-rank matrices. We denote the modified weight matrices after fine-tuning as W', which can be further expressed as:

$$W' = W + \Delta W, \quad \text{with} \quad \Delta W = AB, \quad \text{where} \quad A \in \mathbb{R}^{d \times r}, \quad B \in \mathbb{R}^{r \times d_W}.$$

LoRA essentially introduces the low-rank matrices A and B to approximate the weight matrix change ΔW after the fine-tuning. These low-rank matrices are updated during the fine-tuning process, while the original weight matrix W remains frozen. This strategy significantly reduces the number of trainable parameters from $d \times d_W$ to $r \times (d + d_W)$ because the rank r is typically set to a small value, such as 4 or 8, to balance the trade-off between model capacity and computational efficiency.

3.3.2 Implementation Details

We use exactly the same training and test sets as in §3.2, but the inputs here are the original headlines with the text information. To help Llama-3-8b understand the boundary between two headlines, we concatenate them using the default end of sequence (EOS) token, `<|begin_of_text|>`, placed between them. For example, for the headline pair `The First Headline` and `The Second Headline`, the com-

[13] There are many alternatives for PEFT, like Prefix Tuning (Li and Liang, 2021), Adapter Layers (Houlsby et al., 2019), and BitFit (Zaken et al., 2021). We select LoRA due to its excellent balance between simplicity and efficiency, making it the top choice for fine-tuning.

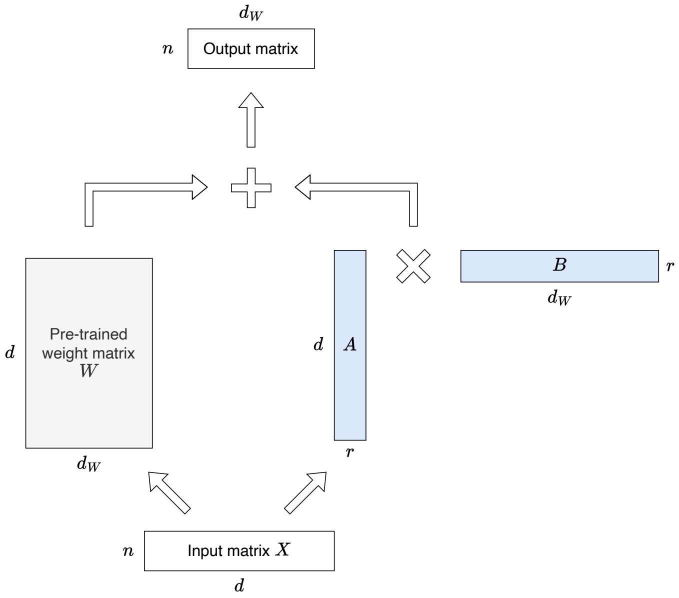


Figure 4: Illustration of LoRA Fine-Tuning.

bined sequence would be, `<|begin_of_text|>The First Headline<|begin_of_text|>The Second Headline`.

We add a classification head to the original Llama-3-8b base model, which includes 32 Transformer layers, as depicted in Figure 5. Llama-3-8b employs a causal attention mask, where the attention is unidirectional—each token can only attend to preceding tokens in the sequence. Thus, the last token in the sequence, being informed by all prior tokens, is used for classification. This embedding vector of the last token (which is 4096-dimensional), output by the last Transformer layer, is processed through a linear layer producing a two-dimensional logit vector, denoted by $Z = (Z_1, Z_2)$. The Z vector is then transformed into probabilities by the softmax function, indicating the likelihood of each headline being catchier, i.e.,

$$\mathbb{P}\{\text{i-th headline is catchier}\} = \frac{e^{Z_i}}{e^{Z_1} + e^{Z_2}} \text{ for } i = 1, 2.$$

During fine-tuning, our objective is to minimize the cross-entropy loss on the training set. We implement LoRA fine-tuning using the PEFT library developed by Hugging Face. We set the dimension of the low-rank matrices as $r = 4$. Although LoRA can be applied to any subset of weight matrices in LLMs, applying LoRA to query weight matrices W_Q and value weight matrices W_V is a common practice and typically gives better performance (Hu et al., 2021). Following this practice, in our experiment, we specifically apply LoRA to the $W_Q \in \mathbb{R}^{d \times d_q}$ and $W_V \in \mathbb{R}^{d \times d_v}$ matrices within the Transformer blocks of Llama-3-8b, where $d = 4096, d_q = 4096, d_v = 1024$. To mitigate over-fitting issues, we also apply a relatively large dropout

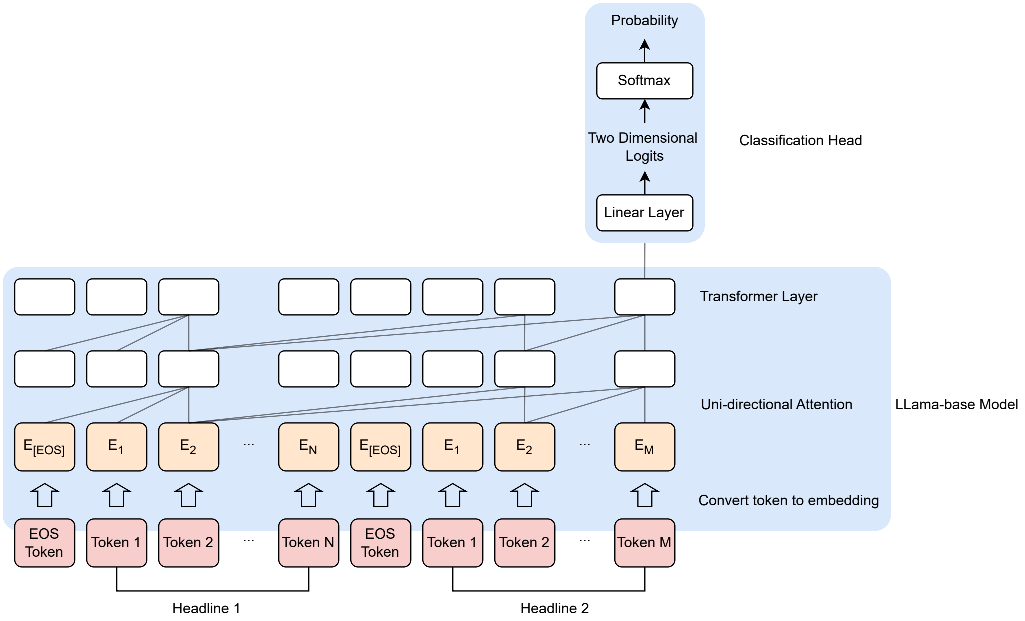


Figure 5: The classification pipeline using the Llama model. Headlines are first tokenized and each token is converted into an embedding vector. For illustrative purposes, only two Transformer layers are shown, although the actual model comprises 32 layers for the 8 billion parameter configuration. The figure highlights the use of uni-directional attention in Llama, whereby each token can only incorporate information from preceding tokens. Connections between Transformer blocks represent the attention mechanism, and only a subset of these links are displayed for simplicity. The output from the last token is fed into a classification head, which computes the probability that one headline is more impactful than the other.

rate of 0.3 to these low-rank matrices.[14]

With this LoRA configuration on Llama-3-8b, the total number of trainable parameters in the model is 1,703,936, which translates to only 0.023% of the total 7,506,636,800 parameters. We fine-tune the model for 10 epochs, meaning each sample (pair of headlines) in the training set is used 10 times in the training. A linearly decreasing learning rate scheduler is employed, starting from an initial rate of 2e-5 and decreasing linearly to zero by the end of the fine-tuning process. This strategy helps stabilize the training by gradually reducing the rate of updates to the weights, thus smoothing the convergence and reducing the risk of overshooting minima in the loss landscape.

3.3.3 Performance Analysis of Fine-tuning-based Approaches

We report the loss curve and accuracy curve during the LoRA fine-tuning process in Figure 6. We observe an accuracy of 83.40% accuracy on the test data, which is marginally better than the best result (82.50% accuracy) using OpenAI embeddings in §3.2. However, such marginal improvements should not undermine the utility of fine-tuning. Comparing the fine-tuning of Llama-3-8b with the use of OpenAI embeddings is not straightforward due to inherent differences in models' capability. To highlight the benefits of fine-tuning,

[14] Furthermore, we enable 16-bit mixed precision training. Although 16-bit training slightly reduces numerical precision, it accelerates operations on GPUs, which enhances computational efficiency significantly. This mode makes the training process faster and consumes less memory compared to the default 32-bit training, especially for large-scale models like Llama-3-8b.

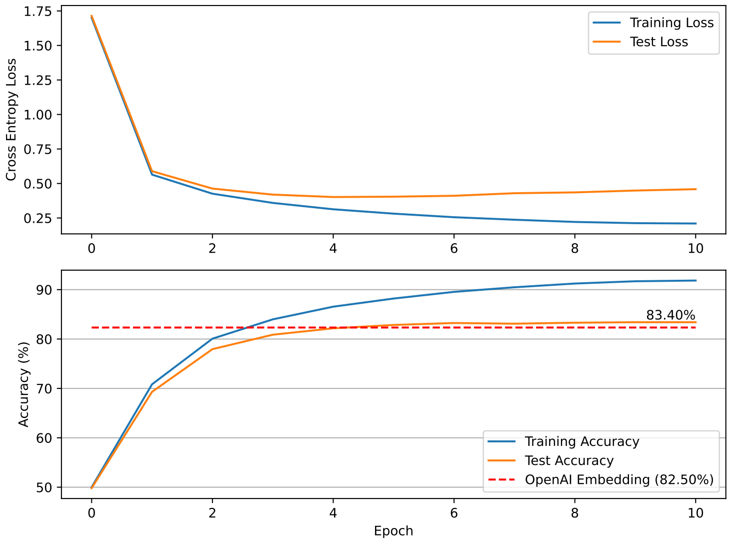


Figure 6: Loss curve and accuracy curve as the number of training epochs increases. We repeat the fine-tuning process three times and display the average loss and accuracy. For each repetition, we consistently used the same training and test sets, ensuring that any randomness in the results came solely from the fine-tuning process, not from variations in the training and test set splits. The final accuracies of the three replicates on the test set are 83.59%, 82.70%, and 83.93% respectively, and 83.40% on average. We add the best performance (red dash line) obtained using OpenAI's embedding in §3.2 for comparison.

we observe that the MLP classification model using embeddings derived from Llama-3-8b (79.40% accuracy in Table 6) performs worse than LoRA fine-tuned Llama-3-8b (83.40% accuracy). This demonstrates that fine-tuning Llama-3-8b can enhance performance compared to Llama-3-8b embeddings.

Similar to the case of the embedding-based classification model, we notice that the fine-tuning approach also suffers from some overfitting issues; see Figure 6. This is because Llama-3-8b is inherently complex; even though LoRA reduces the number of trainable parameters to 0.023% of the total parameters, the model remains complex. Common approaches to address over-fitting include reducing model complexity, adding regularization, and increasing the dataset size. We experimented with various LoRA configurations, such as further reducing the number of trainable parameters, increasing the dropout ratio, and applying larger weight decay in an attempt to mitigate over-fitting. However, these modifications generally reduced performance on the training set without yielding significant improvements on the test set.

We also consider two other versions of the LoRA fine-tuned model to document their performance:

- **Larger dataset:** One way to improve performance and alleviate overfitting is to fine-tune the model on a larger dataset that also includes data on headline pairs that are not significantly different from each other. In our setting, including insignificant pairs of headlines expands the dataset from 39,158 pairs to 134,941 pairs. However, even with this expanded training data, accuracy on the test set continues to hover around 83%, which indicates that incorporating more training data does not necessarily lead to better performance, at least for the current dataset. We refer interested readers to Web Appendix §D.2.1 for details of this analysis and numerical results.

- **DistilBERT Fine-tuning:** Before the widespread adoption of the term "LLM", BERT was considered one of the best language models available, and it is still widely used as a benchmark in many NLP tasks. To provide a comparison with these earlier models, we therefore fine-tune DistilBERT – a distilled version of BERT that retains over 95% of BERT's performance while containing 40% fewer parameters (Sanh et al., 2019; Devlin et al., 2018). Fine-tuned DistilBERT provides an accuracy of about 75% on the test dataset, which is good, but still lower than that of the fine-tuned Llama-3-8b model. Please see Web Appendix §D.2.2 for details on the implementation and performance of the fine-tuned DistilBERT.

In sum, we find that using more data and/or different base models does not lead to significant improvement in performance in our setting.

3.4 Summary of Performance of Pure-LLM Based Methods

We summarize the performance of various pure-LLM-based methods in Table 7. We also compare the accuracy of human respondents on the same task. Details of the survey questions and the accuracy analysis of human respondents are shown in Web Appendix §A.

Method	Accuracy	Notes
Human respondents	51.58%	On 4,571 human labeled pairs
Prompt-based approach	<65%	GPT-4 with in-context learning, evaluated on 1,000 random samples
Classification model with LLM text embeddings	82.50%	OpenAI-3072E embedding model with logistic regression
Fine-tuned LLM model	83.40%	LoRA fine-tuned Llama-3-8b

Table 7: Summarization of performance of various pure-LLM-based methods and comparison with survey results. All results in this table have been reported before. We only report the best performance of each method.

There are four main takeaways from these comparisons. First, human respondents cannot easily predict which headline would be more engaging and likely to lead to higher CTRs. This is consistent with the fact that media firms (and even expert editors) rely on experimentation to decide which headlines and articles to display to users. Second, while simple prompt-based approaches like in-context learning with examples are marginally better than purely random guesses or human predictions, they nevertheless do not provide sufficiently high accuracy. Third, both embedding-based models and fine-tuning perform equally well (at least in our setting), and provide equally good performance with around 83% accuracy. They each have their pros and cons. The main advantage of the logit classification model with OpenAI embeddings is that it is extremely easy to implement and use in business settings since it does not require any significant model training/deployment. However, embeddings from the OpenAI models are proprietary, and as such, we lack visibility into the exact nature of the embeddings. In contrast, fine-tuning approaches use open-source models such as Llama-3-8b, and we have access to the underlying parameters of the LLM. However, this transparency comes with significant costs – fine-tuning LLMs can be computationally expensive and requires non-trivial training and deployment skills that not all businesses may have or want to invest in. The fourth and most important takeaway is that while pure-LLM-based approaches are informative, they are unable to reach the accuracy of experimental approaches. That is, firms cannot replace content experimentation with LLM-based predictions.

Finally, we discuss the monetary costs associated with three LLM-based approaches. The cost of prompt-based, embedding-based, and LoRA fine-tuning in our analysis are around \$20, \$0.18, \$3, respectively. The cost of prompt-based and embedding-based comes from the access to OpenAI's API in order to get responses and text embeddings. The cost of fine-tuning comes from renting a cloud computing server equipped with one NVIDIA A100 GPU. Overall, we find that the prompt-based approaches are the most expensive, while embedding and fine-tuning are the cheapest in terms of cost. However, in terms of run time and effort, fine-tuning is more costly. The detailed comparison of the cost, run-time, and engineering effort required for each pure-LLM method are discussed in detail in Web Appendix §E.

4 Our Approach: LLM-Assisted Online Learning Algorithm

So far, we have seen that pure-LLM-based approaches can help with headline selection with reasonably high accuracy (around 83% for the embedding-based and LoRA fine-tuning approaches). The main advantage of pure LLM-based approaches is that they can be deployed without any experimentation, i.e., the manager can simply use the LLM-based model to predict which headline is better and go with it. Thus, there is no need to use any traffic for experimentation. The downside, of course, is that these models are not fully accurate – in a pairwise comparison, in around 17% of the cases, they are likely to mispredict which headline is better. This can, of course, lead to lower clicks and revenues for the firm.[15] Thus, it is not clear that relying on pure-LLM-based methods will lead to better overall outcomes for the firm.

On the other hand, we have the current status quo, which is the use of experimentation-based approaches to choose headlines/content. As mentioned in §1, there are two broad experimentation-based methods—(1) A/B tests and (2) Online learning algorithms or bandits. In particular, the standard practice at Upworthy during the time of the data collection was A/B tests, where the firm allocated a fixed amount to different headlines and then chose the best-performing one for the remaining traffic. This approach is also known as Explore and Commit (E&C) and is commonly used in the industry due to its simplicity and low engineering cost; see Chapter 6 of (Lattimore and Szepesvári, 2020). The main benefit of this approach is that, with a sufficiently large amount of traffic allocated to the A/B test, the resulting inference is unbiased; as such, it is the gold-standard approach for identifying the best arm (or headline in this case). The downside, of course, is that by assigning traffic to all the headlines during the exploration phase or A/B test, the firm incurs high regret (i.e., low profits since some headlines in the test are likely to have low CTRs). Therefore, many modern technology and media firms such as NYTimes and Yahoo use online learning algorithms or bandits that leverage the explore-exploit paradigm to lower the experimentation cost while still learning (Lattimore and

[15] It should be noted that even this accuracy is calculated under the assumption that the results from the A/B tests are the ground truth. However, even if we were to reach 100% accuracy on this dataset, we still may not reach 100% accuracy in practice because of Type I errors in the creation of significant samples.

Szepesvári, 2020). These algorithms move traffic away from poorly performing arms/headlines dynamically in each period, based on the observed performance of each arm till that period. Because of this greedy behavior, they tend to switch to the best-performing arms quickly and incur lower regret. Nevertheless, one drawback of these methods is that they start with the assumption that all headlines are equally good, i.e., they suffer from a cold-start problem, which can result in wasting traffic on the weaker headlines early on.

In this section, we present a framework that combines the benefits of LLM-based models with experimentation based methods. Our key idea is that firms can address the cold-start problem in content experiments by leveraging predictions from LLM-based approaches as CTR priors in the first step and combining them with an online learning algorithm in the second step. Thus, our approach, termed LLM-Assisted Online Learning Algorithm (LOLA), continuously optimizes traffic allocation based on LLM predictions together with the data collected from the online experiment in real time.

The rest of this section is organized as follows. In §4.1, we present the media firm's problem and the standard bandit framework, and then in §4.2, we present our LOLA framework. Next, in §4.3, we discuss the implementation of our proposed algorithm, benchmarks, and the experimental setup. We present our numerical results in §4.4, and finally, in §4.5, we show how our LOLA approach can be easily generalized to a wide variety of settings and present a set of natural variants and extensions of the basic LOLA framework.

4.1 Bandit Framework

Multi-Armed-Bandits (MAB) is a class of sequential decision-making problems where a learner must choose between multiple arms over periods to maximize cumulative reward. While there is a large literature on it in the machine learning and statistics community, the literature in marketing is small, but growing; researchers have used it for optimizing website design, pricing, advertising, and to learn consumer preferences (Hauser et al., 2009; Schwartz et al., 2017; Misra et al., 2019; Liberali and Ferecatu, 2022; Aramayo et al., 2023; Jain et al., 2024). In our setting, different arms refer to different headlines associated with the same article, and periods refer to impressions where different headlines with the same article are displayed. Let $[K] := \{1, 2, \ldots, K\}$ represent the set of K arms, and $[T] := \{1, 2, \ldots, T\}$ represent the entire decision horizon for a test. At each time step t, the learner selects an arm $a_t \in [K]$ and receives a reward r_{a_t} drawn from a probability distribution specific to the chosen arm a_t. We assume each arm has a constant click-through rate (CTR), and the reward is the click feedback, which follows the Bernoulli distribution with a probability equal to the CTR.

The learner's goal is to minimize cumulative regret, which is defined as the difference between the reward obtained by always choosing the optimal arm and the reward actually accumulated by the learner under the policy played by the learner. Mathematically, the regret R_T after T trials is given by:

$$R_T = \max_{k \in [K]} \sum_{t=1}^{T} r_{a^*} - \sum_{t=1}^{T} r_{a_t}, \tag{1}$$

where $a^* = \arg\max_{k \in [K]} \mathbb{E}[r_k]$ is the optimal arm with the highest expected reward.

In MAB settings with finite arms, the standard algorithm that is typically used is the Upper Confidence Bound (UCB) algorithm (Auer et al., 2002). UCB stands out among all online learning algorithms because it is provably asymptotically optimal. The UCB policy is straightforward – it selects the arm that maximizes the upper confidence bound on the estimated rewards. Specifically:

$$a_t = \arg\max_{k \in [K]} \left(\bar{\mu}_k^t + \sqrt{\frac{2\log(1/\delta)}{n_k^t}} \right), \tag{2}$$

where $\bar{\mu}_k^t$ is the estimated mean reward of arm k up to time t, n_k^t is the number of times arm k has been played up to time t, and δ is the confidence level, which quantifies the degree of certainty. The intuition behind UCB's effectiveness in balancing exploration and exploitation lies in its construction. By adding a confidence term $\sqrt{2\log(1/\delta)/n_k^t}$ to the estimated mean reward $\bar{\mu}_k$, UCB ensures that arms with fewer selections (higher uncertainty) are given a chance to be explored. As time progresses and more data is collected, the confidence term diminishes, allowing the algorithm to exploit arms with higher observed rewards. This dynamic adjustment enables the UCB algorithm to explore under-explored arms while exploiting arms with high rewards systematically, thus effectively addressing the exploration-exploitation trade-off inherent in the MAB problem. However, we need to set the key hyperparameter δ such that it is small to ensure optimality with high probability, but not so small that suboptimal arms are over-explored; see Chapter 7.1 (Lattimore and Szepesvári, 2020) for a detailed discussion. Typically, we set the confidence term as $\alpha\sqrt{\log t/n_k^t}$ and fine-tune the scale term α for specific problems.

4.2 LOLA

We now discuss our approach LOLA, and how the standard UCB approach discussed above can be extended to include information from LLMs in the first step preceding active experimentation. LOLA has two key steps, and we describe each in detail.

In the first step, we use LLMs to get an initial prediction of each headline's CTR. Note the prediction task here is somewhat different from that in §3. In the previous section, we used LLM-based approaches to perform pairwise comparisons of headlines and predict which headline amongst a pair is more engaging. However, as discussed earlier, that approach does not work when we have more than two headlines (which is the case for over 90% of the A/B tests in our data). Further, we need actual CTR predictions for each headline to feed into the second step of our algorithm. Therefore, we adapt the LLM-based classification models from §3 for predicting CTRs by making the following modifications: changing the loss function from log loss to mean squared error (MSE), using single headline inputs instead of headline pairs, and switching from binary output to continuous CTR prediction. The training and fine-tuning procedures remain unchanged.[16]

We compare two approaches for CTR prediction: an OpenAI-embedding-based method and a LoRA

[16] Using CTR as the outcome measure does not adversely affect the performance. For instance, using linear regression with 256-dimensional text embeddings from OpenAI to predict CTR for significant pairs and labeling the headline with the higher predicted CTR as the winner yields an accuracy of around 75%, which is very close to the 77.31% accuracy obtained with logistic regression. This minor difference is likely due to the inherently skewed nature of the CTR distribution and the sensitivity of MSE to outliers.

Algorithm 1 LLM-Assisted 2-Upper Confidence Bounds (LLM-2UCBs)

```
**LLM Training Phase:** Train a LLM-based prediction model $\mathcal{M}(x)$ for CTRs using historical data samples, where $x$ is the contextual information for arms.
**Hyperparameter Fine-Tuning:** Use another subset of data to select two hyperparameters for the algorithm, $\alpha \in \mathbb{R}^+$ for controlling the upper bound, $n^{\text{aux}}$ as the LLM's equivalent auxiliary sample size.
**Online Learning Phase:** Initialize the number of periods $T$, number of arms $K$, LLM-based CTR prediciton $\bar{\mu}_k^{\text{aux}} \leftarrow \mathcal{M}(x_k)$, regular CTR initialization $\bar{\mu}_k^1 = 0$, accumulated impressions $n_k^1 \leftarrow 1$, and accumulated clicks $c_k^1 \leftarrow 0$, for all arms $k \in [K]$.
**for** $t = 1$ **to** $T$ **do**
    Calculate the first UCB for all arms $k \in [K]$: $U_k^1 = \bar{\mu}_k^t + \alpha\sqrt{\frac{\log t}{n_k^t}}$.
    Calculate the second UCB for all arms $k \in [K]$:
        $$U_k^2 = \frac{c_k^t + \bar{\mu}_k^{\text{aux}} n^{\text{aux}}}{n_k^t + n^{\text{aux}}} + \alpha\sqrt{\frac{\log t}{n_k^t + n^{\text{aux}}}}.$$
    Play the arm $a_t = \arg\max_{k \in [K]} \min\{U_k^1, U_k^2\}$.
    Observe the payoff $r_{a_t}$
    Update $n_{a_t}^{t+1} \leftarrow n_{a_t}^t + 1$, $c_{a_t}^{t+1} \leftarrow c_{a_t}^t + r_{a_t}$, and $\bar{\mu}_{a_t}^{t+1} \leftarrow \frac{c_{a_t}^{t+1}}{n_{a_t}^{t+1}}$.
**end for**
```

fine-tuned Llama-3-8b model. Our numerical results indicate that the R-squared values for the embedding-based approach and the LoRA fine-tuned method are 0.241 and 0.315, respectively. When selecting the headlines with the highest predicted CTRs, the accuracy—defined as the percentage of times the chosen headline indeed has the highest actual CTR—is 42.66% for the embedding-based method and 46.55% for the LoRA fine-tuned method. Note that this accuracy is significantly lower than the binary classification accuracy discussed in §3 because we can have more than two headlines in a test and many headlines have no significant difference in CTRs. Based on these results, we select the LoRA fine-tuned Llama-3-8b model as the best CTR prediction model in the first step. Additional details on the CTR prediction methods and their results can be found in Web Appendix §F.

In the second step, we build on the bandit framework from §4.1 to design an online algorithm that incorporates the LLM-based CTR predictions from the first step. Inspired by the 2-Upper Confidence Bounds (2-UCBs) policy designed by Gur and Momeni (2022), we propose Algorithm 1, titled LLM-Assisted 2-Upper Confidence Bounds (LLM-2UCBs). Essentially, in this algorithm, we treat the LLM's CTR predictions as auxiliary information prior to the start of online experiments, equating this auxiliary CTR information to additional impressions and click outcomes for each arm.

There are two hyperparameters in LLM-2UCBs – (1) The upper bound control hyperparameter, α, that also shows up in the regular UCB algorithm, and (2) The hyperparameter n^{aux}, which is specific to our proposed algorithm and indicates the equivalent auxiliary samples for initialization. At a high level, we can regard the LLM predictions as prior information or the information flow before the algorithm starts. This prediction is approximately equivalent to initially generating n^{aux} impressions for each arm and observing their outcomes to obtain the sample average CTR, $\bar{\mu}_k^{\text{aux}}$. The hyperparameter fine-tuning process is simple.

We can utilize another randomly sampled dataset to test combinations of two hyperparameters and select the combination that yields the highest accumulated rewards.

During the online learning phase, the key step is balancing the exploration and exploitation trade-off. We capture this trade-off using the 2-UCBs rule. Essentially, we have two valid UCBs: U_k^1 is the regular UCB, which uses observed impressions and clicks to construct the mean reward estimator and adds a regular upper bound. The second UCB, U_k^2, incorporates LLM predictions. The calculation of U_k^2 is straightforward. The mean reward component is simply the sample average, taking into account the "auxiliary" samples generated by the LLM before the learning phase begins. These auxiliary samples are imaginary and approximate the richness of information contained in the LLM prediction. The next step is to choose the smaller of the two UCBs, $\min\{U_k^1, U_k^2\}$, as the final UCB for selecting the arm. Note that if $n^{\text{aux}} = 0$, the 2-UCBs rule is the same as the standard UCB rule. Conversely, when $n^{\text{aux}} \to \infty$, the 2-UCBs rule is equivalent to a pure exploitation policy (a.k.a. greedy policy) using the LLM-based CTR prediction. Thus, when n^{aux} is a positive integer, the 2-UCBs algorithm incorporates information from both LLMs and the actual click outcomes. After playing the arm and observing the outcome, we update the impressions, clicks, and mean reward accordingly, following standard bandit algorithm procedures.

The 2-UCBs algorithm offers three advantages. First, it is easy to implement, requiring the tuning of only two hyperparameters. Second, it can be used in conjunction with any LLM-based CTR prediction model in the first step and any outcome of interest in the second step. We use clicks as the outcome of interest given our application setting and data; however, other outcomes, such as time spent on a page or other measures of engagement, can be easily used instead of clicks. Finally, it is intuitive and easy to implement. Because it can easily accommodate the edge cases, e.g., one where the manager has no auxiliary information or one where the manager wants to fully rely on the LLM predictions, there is no need for extra engineering effort to set up separate systems for different edge cases. This adaptability is extremely valuable given the rapid advances in LLM technology. When the capability of the pure-LLM method continues to advance, the manager can easily fine-tune and increase n^{aux} to take more advantage of LLM.

4.3 Benchmarks and Implementation Details

We now discuss the implementation details and present a set of benchmark algorithms. First, we randomly split our data into three exclusive folds based on the test ID, ensuring that all headlines in the same test are in the same fold and that the same headline does not appear in different folds. We use 70% of the data for training, 10% for hyperparameter fine-tuning, and 20% for testing. The training data is used to train the LLM models and is only used by our proposed LOLA approach (the LLM-2UCBs specifically) and pure LLM-based benchmarks. All hyperparameters in all the algorithms (both LOLA and the benchmark algorithms) are selected using the fine-tuning data, and the performance of the algorithms is evaluated on the test data.

We now describe a set of commonly used algorithms that serve as benchmarks and our LOLA approach.

- **Explore-then-Commit (E&C):** In this approach, the firm runs the A/B test for a fixed number of periods (i.e., explores) and then commits to the best-performing arm at that stage; see Chapter 6 (Lattimore and

Szepesvári, 2020). This is the standard A/B test approach used by many digital firms in the industry. In particular, Upworthy's approach during the data collection period closely resembles the E&C algorithm – recall that they assign traffic to candidate headlines with equal probability during the experiment phase and then show the winning headline to all future impressions. The E&C method has a single hyper-parameter that needs tuning: the proportion of periods for exploration. We consider the following proportions – $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, and choose the one with the highest accumulated rewards based on the 10% of data used for hyper-parameter tuning. We finally choose 0.2 for exploration.

- **Pure LLM-based approach:** In this approach, the firm uses LLMs to predict the CTR for all the arms (headlines) and then allocates all the traffic to the arm with the highest predicted CTR. This is equivalent to a fully greedy policy based on LLM predictions. We use the training data to learn a model that predicts the CTR of a headline based on an LLM (either through text embeddings or LoRA fine-tuning). In our numerical study, we use the best-performing pure-LLM-based approach based on our experiments (see Web Appendix §F), i.e., the LoRA fine-tuned Llama-3-8b with MSE loss, which gives a 46.55% accuracy.[17]
- **Standard UCB (UCB):** This is the standard UCB algorithm discussed in §4.1. Here, the firm does not use any first-stage LLM predictions as an input, and there is only one hyperparameter to fine-tune, the confidence control, α. We consider values from the set $\{0.02, 0.04, \ldots, 0.10\}$, and find that $\alpha = 0.08$ gives the highest accumulated rewards.
- **LOLA:** This is our proposed approach. In the first step, we use CTR predictions from the best-performing pure-LLM-based CTR prediction model, i.e., the LoRA fine-tuning Llama-3-8b with MSE loss. Note that this is the same model that we use for the pure LLM-based approach, which allows for a fair comparison of the two approaches. For the second step, we use the LLM-2UCBs algorithm as outlined in §4.2. We fine-tune the combination of the two hyperparameters on the fine-tuning data, and we consider $n^{\text{aux}} \in \{600, 800, 1000, 1200, 1400\}$ and $\alpha \in \{0.02, 0.04, \ldots, 0.10\}$. Based on fine-tuning, we choose $n^{\text{aux}} = 1000$ and $\alpha = 0.08$ for our setting.

  Note that the former two methods can be viewed as special cases of our proposed LOLA algorithm. Both our algorithm and the pure LLM-based approach use the exact same CTR prediction algorithm. Hence, the pure LLM-based approach (a.k.a pure exploitation, greedy algorithm) is a special case of LOLA, wherein we set $n^{\text{aux}} = \infty$, indicating complete trust in the LLM prediction results. This method fully relies on the LLM CTR prediction, directly selecting the headline with the highest predicted CTR throughout the entire horizon. Similarly, the standard UCB algorithm is a special case of LOLA with $n^{\text{aux}} = 0$, indicating no reliance on LLM CTR predictions. Again, note that we use the same hyper-parameter $\alpha = 0.08$ across both algorithms, which ensures that our algorithm defaults to the standard UCB with $\alpha = 0.08$ when we ignore the CTR predictions from the LLM.

After the fine-tuning of algorithms, we evaluate all algorithms on the test dataset, which has never been

[17] Note that since this is a pure prediction-based greedy approach, there are no hyper-parameters associated with the algorithm, and hence we do not use the fine-tuning data.

used for either CTR training or algorithm tuning. Therefore, none of the algorithms know the true CTRs of the headlines in the test data in advance. We use the average CTR for each headline in the test data as the true CTRs to generate the click feedback in our numerical simulations. This approach of generating click outcomes is reasonable in our setting since each headline was tested on a large amount of traffic by Upworthy, resulting in relatively narrow confidence intervals.

UCB and LOLA are both asymptotically optimal in minimizing regret, which means there is no difference between these two algorithms when the number of time horizons T goes to infinity, and the LLM-based prior information is not helpful in an infinite horizon case. However, in practice, the time horizon is never infinite. Therefore, we compare all the algorithms for different values of time horizons. One challenge in our setting is that the number of headlines varies across tests. Intuitively, for any given time horizon, it is always easier to minimize regret (or identify the best headline) when the A/B test has only two headlines compared to a case when there are three headlines; and, of course, the problem only gets more challenging as the number of headlines increases. As a result, comparisons across time horizons are meaningful only when we keep the number of headlines constant. Since the number of headlines varies in our dataset (see Table 3), we scale the time horizon for each A/B test using the metric "Traffic/Impressions per headline" (which we represent using the notion τ), where we choose τ from $\{50, 100, 200, 400, 600, 800, 1000\}$. For example, if there are three headlines in an A/B test and the multiplier is $\tau = 100$, then we set $T = 100 \times 3 = 300$ for this test. Thus, the effective horizon across all tests for a given τ is the same. In our experiments, it will be particularly important to investigate the performance of different algorithms at small-medium multipliers because these represent situations where the media firm only has a small amount of traffic to work with and needs to adaptively experiment and effectively allocate impressions to the right headline with limited experimentation ability.

4.4 Numerical Results

We visualize the results from our numerical simulations in Figure 7 and document the pairwise comparison of different algorithms, along with the relative percentage reward improvement and the p-values from the t-tests in Table 8. We find that, on average, LOLA performs the best among all algorithms across all time horizons. We document a few additional patterns of interest.

First, the Pure LLM approach exhibits consistent performance across different τ values because it is not an adaptive algorithm. The small fluctuation at $\tau \in \{50, 100, 200\}$ is due to the randomness of average clicks per period, which tends to be larger under the smaller time horizon. For the other three approaches, which actively learn the CTR during the experiments, their average performance per period increases as the time horizon extends. This improvement is because the number of arms in the experiment remains constant, and with increasing periods, these algorithms have more opportunities to learn the CTRs, thus achieving better performance. Comparing the performance of LOLA with pure LLM methods, we find that LOLA generally outperforms Pure LLM, except when $\tau = 50$, where there is no significant difference between LOLA and Pure LLM. This is because the short planning horizon does not allow LOLA to learn new information over and above the information already available from the LLM. As τ increases to 1000, we see that LOLA significantly outperforms Pure LLM, generating 5.02% more clicks. This suggests that if the horizon is

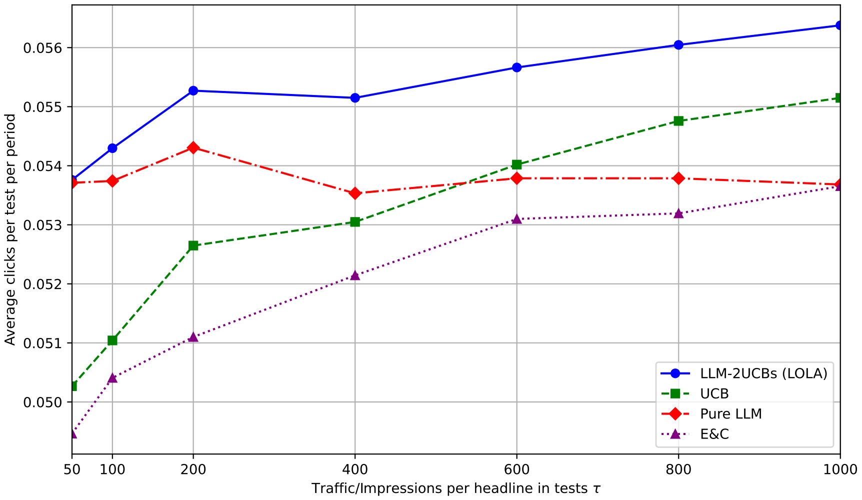


Figure 7: Average clicks per experiment per period under different time horizon multipliers. Note that the Y-axis captures the average clicks per test per period. For instance, if there is a test with two headlines conducted under $x = 100$, then the average click per period in this test is calculated as $(1+2)/100 = 0.03$. The Y value is simply the average of this number 0.03 over all tests. This measure scales well with the platform's total clicks in tests because headlines in different tests with different numbers of headlines take the same weight in this measure.

Parameter τ	LOLA vs UCB	LOLA vs Pure LLM	LOLA vs E&C	UCB vs Pure LLM	UCB vs E&C	Pure LLM vs E&C
50	6.95^{****}	0.09	8.69^{****}	-6.41^{****}	1.62	8.59^{****}
100	6.38^{****}	1.03^{**}	7.72^{****}	-5.02^{****}	1.26	6.61^{****}
200	4.98^{****}	1.78^{****}	8.16^{****}	-3.05^{****}	3.03^{**}	6.27^{****}
400	3.96^{****}	3.02^{****}	5.76^{****}	-0.90	1.74^{*}	2.66^{***}
600	3.04^{****}	3.49^{****}	4.83^{****}	0.43	1.73^{**}	1.30
800	2.35^{****}	4.20^{****}	5.36^{****}	1.81^{***}	2.94^{****}	1.12
1000	2.23^{****}	5.02^{****}	5.08^{****}	2.73^{****}	2.79^{****}	0.05

Table 8: Percentage % reward improvement between two algorithms with significance levels: * $p \leq 0.05$, ** $p \leq 0.01$, *** $p \leq 0.001$, **** $p \leq 0.0001$. For example, under the impression per headline equal to 50, the average click per test of LOLA and UCB is 0.053760 and 0.050267, respectively; so the relative improvement is calculated as $0.053760/0.050267 - 1 = 6.95\%$. Parameter τ represents the average impression per headline to scale the time horizon $T = \tau \times K$.

very short, the planner/firm will not see significant improvement over simply relying on LLM predictions. However, as the horizon increases, LOLA will start showing significant gains.

Next, comparing LOLA with regular UCB, we see that it always outperforms UCB. Interestingly, here we see that, as the time horizon increases, the relative improvement from LOLA diminishes. Intuitively, this is because both UCB and LOLA are active learning algorithms, and as the time horizon increases, they are both able to learn the CTRs better. While LOLA starts with a relative information advantage, as the time horizon increases, this advantage diminishes. Formally, this pattern is due to the asymptotic optimality of the UCB and LOLA algorithms, implying that as the time horizon approaches infinity, the initialization has a negligible effect on the average performance over an infinitely long period. We also see that E&C consistently underperforms compared to LOLA because it neither leverages LLM predictions nor achieves asymptotic optimality. While E&C shows better performance as the horizon increases, even when $\tau = 1000$, LOLA is still better than E&C by 5.08%, which is a large improvement in the media and publishing industry.

Finally, comparing the benchmark algorithms with each other, we see that when the horizon is short, the Pure LLM does better, while UCB is better when the horizon is longer. In general, this reflects the intuition that active learning is not very useful when the time horizons are very short since the algorithm has no time to explore and learn the CTRs effectively (and then exploit this learning). So simply going with the pure LLM predictions is better. In contrast, when the horizons are longer, UCB (or active learning) outperforms using static predictions from the LLM. Note that LOLA combines the strengths of both UCB and Pure LLM, and hence it is able to outperform both and provide superior performance in both short and long horizons. We also note that E&C is generally worse than both Pure LLM and UCB, which suggests that the current Upworthy practice is suboptimal. This is because E&C neither exploits LLM information and nor does it actively learn over time.

In summary, we find that LOLA outperforms existing approaches for experimentation by leveraging the strengths of LLMs and combining them with the benefits of active/online learning. In particular, our results demonstrate the value of LOLA as the time horizon increases, and highlight the limitations of relying solely on pure LLM predictions or the naive E&C approach.

4.5 LOLA Variants and Extensions

So far, we have considered a version of LOLA that uses a fine-tuned CTR prediction model for LLM predictions in the first stage and an LLM-2UCBs algorithm that minimizes regret for online learning in the second step. However, LOLA is a general framework and can be easily adapted to a wide variety of settings. We highlight a few natural extensions and variants below. Details of the models and numerical results (when applicable) are shown in Web Appendix §G.

Best Arm Identification: So far, we focused on the goal of regret minimization, where the goal is to maximize the clicks/reward (per Equation (1)). While regret minimization is the most commonly studied goal in active learning and the natural goal for businesses in most settings, another commonly studied goal is Best Arm Identification (BAI). The goal of BAI problems is to identify the arm with the highest reward as fast and accurately as possible, regardless of accumulated regret. Formally, given a low failure rate δ, we aim to find the arm a^* with the highest expected reward, at least $1 - \delta$ probability, while minimizing the number of total pulls.[18] We can easily modify the LOLA framework for a BAI goal. In Web Appendix §G.1, we present a LLM-BAI algorithm that builds on the recently proposed BAI algorithm by Mason et al. (2020). This algorithm is particularly effective in our setting since it can accommodate scenarios where the difference between the rewards of arms is small or insignificant. We show that our LLM-BAI algorithm significantly outperforms both the plain BAI algorithm (proposed by Mason et al. (2020)) as well as the standard A/B test under the same failure rate restriction. Please see Web Appendix §G.1 for a detailed description of the

[18] This is also called fixed-confidence setting (Garivier and Kaufmann, 2016). Another common approach is the fixed-budget setting, where the goal is to maximize the probability of finding the best arm within a fixed number of pulls.

LLM-BAI algorithm and its performance against the standard benchmarks.

Thompson Sampling: Our bandit specification in §4.1 reflects a stochastic bandit setting rather than a Bayesian bandit setting. However, our proposed algorithms can easily be applied in a Bayesian setting. We present a Thompson sampling version of our proposed LOLA algorithm (LLM-TS) and its numerical performance in Web Appendix §G.2. We see that UCB-based algorithms perform better than Thompson Sampling-based algorithms in both the LLM-assisted version and the standard version. That is, LLM-TS performs worse than LLM-2UCBs, and standard TS performs worse than UCB. This pattern has been observed in the literature, and it is generally known that TS suffers from over-exploration (Min et al., 2020); in other words, UCB's upper confidence shrinks faster, leading it to exploit more effectively. Nevertheless, in business settings where the firm already has a TS-based adaptive experimentation framework, it is easy to adapt the LOLA approach within this approach.

User Information: So far, we have not considered user-level features or how to personalize the content shown to different users based on their behavioral/context features (mostly because the UpWorthy data does not have user-level features). Prior research has shown that using such features can significantly improve the match between users and content (Li et al., 2010; Yoganarasimhan, 2020; Rafieian and Yoganarasimhan, 2021). It is, however, easy to expand our LOLA approach to include user features in both the LLM training phase as well as the online learning phase. We present two versions of LOLA that extend existing contextual bandit algorithms to our setting. First, we can extend the standard contextual linear bandits (Chu et al., 2011) to our LOLA framework by incorporating both user and text features in the LLM training and online learning phases of LOLA. However, this linear bandit approach has limitations since it assumes that the rewards model is linear in text and user features. Therefore, we also propose another solution based on the FALCON algorithm that does not make any linearity assumption (Simchi-Levi and Xu, 2022). We present the details of both these contextual LOLA algorithms and their pros and cons in Web Appendix §G.3.

Alternative rewards: In the Upworthy setting, the measure of reward is clicks. However, click-based metrics have raised concerns about promoting clickbait headlines, which may reduce user engagement or spread negativity. As a result, the news and media industry is now evolving to focus more on content quality and humanity, although clicks still remain an important measure; see Reneau (2023) for a more detailed discussion. The LOLA framework can be easily adapted to alternative reward measures that align with new business goals, e.g., time spent on the platform during a session, or time spent on the article (and not just clicks on the headline).

Apart from these extensions, the LOLA approach is quite general and adaptable on other dimensions as well. For instance, it is agnostic to the exact nature of the LLM-based approach used in the first step. Any of the LLM-based approaches discussed in §3 can be used in the first step. This is important since the LLM models and fine-tuning approaches continue to advance quickly. Further, LOLA can also be used in conjunction with content/creatives created by LLMs themselves. For instance, one simple way to leverage fine-tuned models is to use them to generate content (instead of using human-generated headlines/content). Indeed, recent studies have shown that promotional content (ads/emails) generated by fine-tuned LLMs tend to outperform traditional personalized ads and human-generated content in effectiveness (Kumar and Kapoor, 2023; Angelopoulos et al., 2024). LLM-generated headlines/content can be used as competing arms/treatments within our LOLA framework. Additionally, the LOLA framework can also be easily extended to content recommendation systems used by social media platforms such as X and Facebook. In summary, the plug-and-play nature of our approach can easily accommodate newer advances and developments at all stages of the implementation process.

5 Conclusion

In conclusion, in this paper, we examine if and how firms can leverage the LLMs to enhance content experimentation in digital platforms. First, we examine how well LLMs can predict which content would be more appealing to users. We find that while LLMs provide informative predictions, even the best performing fine-tuned LLMs are unable to perfectly predict which content will be more appealing to users, and match the accuracy/regret of experimentation-based approaches. As such, firms cannot fully trust the prediction results solely from LLMs and replace experimentation.

We introduce LOLA, a novel experimentation framework that combines the predictive power of LLMs with the adaptive efficiency of online learning algorithms to enhance content experimentation. By leveraging LLM-based prediction models as priors, our approach minimizes regret in real time, leading to significant improvements in total reward. Comprehensive numerical experiments based on a large-scale A/B testing dataset demonstrate that LOLA outperforms the traditional A/B tests and pure online learning algorithms.

From a managerial perspective, LOLA offers a versatile solution to a broad set of scenarios where the firm needs to decide which content to display to users. LOLA is particularly valuable under limited traffic conditions that are common in many digital media platforms. This includes cases where content becomes stale quickly (e.g., news articles) or social media platforms like X and TikTok, where users generate a lot of new content regularly, and as a result, the amount of content is high relative to the amount of traffic. While our focal application is in the context of headline selection in the news and media industry, the framework can easily be applied to other problems, such as digital advertising, email marketing, and promotions. Indeed, a large literature in marketing focuses on how to use experiments in conjunction with machine learning methods to identify optimal treatments and/or personalize treatments; see Rafieian and Yoganarasimhan (2023) for a detailed review. The LOLA approach can be easily used in many of those settings and help with maximizing outcomes of interest while simultaneously reducing the cost of experimentation. Finally, from an implementation perspective, LOLA offers many advantages. Since it can utilize open-source LLM models, it is cost-effective and can be easily deployed across different tasks once LLM models are fine-tuned on relevant datasets. The fine-tuning process itself is cost-efficient and can be performed on entry-level GPUs, making advanced LLM-based techniques accessible to firms with limited budgets. Relying on open-source LLMs also ensures that the firm's data stays within the firm and is not accessible to the owners of proprietary LLM models (such as Google and OpenAI). Further, our approach is compatible with many bandit-based experimental systems used by large digital firms such as Amazon (Fiez et al., 2024), and as such, is easy to integrate and deploy.

Code and Data

The code used in this study is publicly available at LLM News Github Repository to enable the reproduction of our results and to support future research that applies LLMs to experimentation and business problems.

Funding and Competing Interests Declaration

Author(s) have no competing interests to declare.

References

Panagiotis Angelopoulos, Kevin Lee, and Sanjog Misra. Value aligned large language models. *Available at SSRN 4781850*, 2024.

Nicolás Aramayo, Mario Schiappacasse, and Marcel Goic. A multiarmed bandit approach for house ads recommendations. *Marketing Science*, 42(2):271–292, 2023.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*, 2024.

James Brand, Ayelet Israeli, and Donald Ngwe. Using gpt for market research. *Available at SSRN 4395751*, 2023.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

Anna Coenen. How The New York Times is experimenting with recommendation algorithms. *NYT Open*, 2019. URL https://open.nytimes.com/how-the-new-york-times-is-experimenting-with-recommendation-algorithms-562f78624d26. Accessed: 2024-05-24.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Tanner Fiez, Houssam Nassif, Yu-Cheng Chen, Sergio Gamez, and Lalit Jain. Best of three worlds: Adaptive experimentation for digital marketing in practice. In *Proceedings of the ACM on Web Conference 2024*, pages 3586–3597, 2024.

Aurélien Garivier and Emilie Kaufmann. Optimal best arm identification with fixed confidence. In *Conference on Learning Theory*, pages 998–1027. PMLR, 2016.

George Gui and Olivier Toubia. The challenge of using LLMs to simulate human behavior: A causal inference perspective. *arXiv preprint arXiv:2312.15524*, 2023.

Yonatan Gur and Ahmadreza Momeni. Adaptive sequential experiments with unknown information arrival processes. *Manufacturing & Service Operations Management*, 24(5):2666–2684, 2022.

John R Hauser, Glen L Urban, Guilherme Liberali, and Michael Braun. Website morphing. *Marketing Science*, 28(2):202–223, 2009.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Lalit Jain, Zhaoqi Li, Erfan Loghmani, Blake Mason, and Hema Yoganarasimhan. Effective adaptive exploration of prices and promotions in choice-based demand models. *Marketing Science*, 2024.

Madhav Kumar and Anuj Kapoor. Generative AI and personalized video advertisements. *Available at SSRN 4614118*, 2023.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

Peiyao Li, Noah Castelo, Zsolt Katona, and Miklos Sarvary. Frontiers: Determining the validity of large language models for automated perceptual analysis. *Marketing Science*, 2024.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

Gui Liberali and Alina Ferecatu. Morphing for consumer dynamics: Bandits meet hidden markov models. *Marketing Science*, 41(4):769–794, 2022.

Blake Mason, Lalit Jain, Ardhendu Tripathy, and Robert Nowak. Finding all ϵ-good arms in stochastic bandits. *Advances in Neural Information Processing Systems*, 33:20707–20718, 2020.

J Nathan Matias, Kevin Munger, Marianne Aubin Le Quere, and Charles Ebersole. The upworthy research archive, a time series of 32,487 experiments in US media. *Scientific Data*, 8(1):195, 2021.

Meta. Llama 3 model card, 2024a. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Meta. Introducing Meta-Llama 3, 2024b. URL https://ai.meta.com/blog/meta-llama-3. Accessed: 2024-05-28.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Seungki Min, Ciamac C Moallemi, and Daniel J Russo. Policy gradient optimization of thompson sampling policies. *arXiv preprint arXiv:2006.16507*, 2020.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint* *arXiv:2202.12837*, 2022.

Kanishka Misra, Eric M Schwartz, and Jacob Abernethy. Dynamic online pricing with incomplete information using multiarmed bandit experiments. *Marketing Science*, 38(2):226–252, 2019.

Rajvardhan Patil, Sorio Boit, Venkat Gudivada, and Jagadeesh Nandigam. A survey of text representation and embedding techniques in NLP. *IEEE Access*, 2023.

Omid Rafieian and Hema Yoganarasimhan. Targeting and privacy in mobile advertising. *Marketing Science*, 40(2):193–218, 2021.

Omid Rafieian and Hema Yoganarasimhan. Ai and personalization. *Artificial Intelligence in Marketing*, pages 77–102, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Annie Reneau. Study of upworthy headlines claims negativity drives website clicks. We have some thoughts. *Upworthy*, 2023. URL https://www.upworthy.com/upworthy-negative-headlines-study. Accessed: 2024-05-24.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. A mathematical exploration of why language models help solve downstream tasks. *arXiv preprint arXiv:2010.03648*, 2020.

Eric M Schwartz, Eric T Bradlow, and Peter S Fader. Customer acquisition via display advertising using multi-armed bandit experiments. *Marketing Science*, 36(4):500–522, 2017.

David Simchi-Levi and Yunzong Xu. Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability. *Mathematics of Operations Research*, 47(3):1904–1931, 2022.

Sibel Sozuer Zorlu, Oded Netzer, and Kriste Krstovski. A recipe for creating recipes: An ingredient embedding approach. *Available at SSRN 4686749*, 2024.

Alexandria Symonds. When a headline makes headlines of its own. *The New York Times*, 2017. URL https://www.nytimes.com/2017/03/23/insider/headline-trump-time-interview.html. Accessed: 2024-05-24.

Artem Timoshenko and John R Hauser. Identifying customer needs from user-generated content. *Marketing Science*, 38(1):1–20, 2019.

Xin Wang, Jiaxiu He, David J Curry, and Jun Hyun Ryoo. Attribute embedding: Learning hierarchical representations of product attributes from consumer reviews. *Journal of Marketing*, 86(6):155–175, 2022.

Sang Michael Xie and Sewon Min. How does in-context learning work? A framework for understanding the differences from traditional supervised learning. *SAIL Blog*, 2022. URL http://ai.stanford.edu/blog/understanding-incontext/.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

Kathy Yang. Milestones on our journey to standardize experimentation at The New York Times. *NYT Open*,

2024. URL https://open.nytimes.com/milestones-on-our-journey-to-standardize-experimentation-at-the-new-york-times-2c6d32db0281. Retrieved: 2024-05-24.

Hema Yoganarasimhan. Search personalization using machine learning. *Management Science*, 66(3): 1045–1070, 2020.

Hema Yoganarasimhan and Irina Yakovetskaya. From feeds to inboxes: A comparative study of polarization in facebook and email news sharing. *Management Science, forthcoming*, 2024.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. LoRA land: 310 fine-tuned LLMs that rival GPT-4, a technical report. *arXiv preprint arXiv:2405.00732*, 2024.

Web Appendix

A User Survey

We conduct a survey of undergraduate students from a large public university. The goal of the survey was to examine whether lay human users can accurately identify which headlines would lead to more clicks. Recall that the headlines themselves were created by Upworthy editors, but those editors need an experiment to identify the most appealing headline.

We design two similar surveys using Qualtrics. The first survey randomly samples 3,000 pairs of headlines from 39,158 headline-pairs that are significantly different from each other (at a 0.05 significance level), along with 500 pairs that have no significant differences in CTRs. The second survey utilizes the same 3,000 significant samples as the first but does not include the insignificant samples.

In both surveys, students were asked to select the catchier headline from a pair of headlines. The introductory text for the surveys was:

"Please help us understand what makes a news headline appealing to readers. In this survey, you will be presented with 20 pairs of headlines. Please choose the one that captures your attention more effectively or feels more engaging in each pair, assuming both headlines relate to the same news content. There are no right or wrong answers; we are simply interested in your opinions."

Respondents then proceed to a new page featuring 20 headline comparison questions with the prompt, "Please select the headline that you find more attractive," with only one permissible response. In all the comparisons, we randomly swap the order of the headlines to ensure that the display order does not correlate with the correct answer. We also incorporated two attention checks in the second survey.

The first survey included an additional response option: "Both headlines seem equally good." At the survey's conclusion, we also collected data on users' news consumption habits. Specifically, we ask, "How often do you read the news?" with options "Never", "1-2 days per week", "3-4 days per week", "5-6 days per week", "Everyday".

Detailed information on the surveys and the results from both surveys are shown in Table A1. Note that the total responses received is less than the number of respondents ×20 because some students did not finish the survey or skipped some comparisons. For example, for the first survey, the total number of responses 3,809 is less than $202 \times 20 = 4,040$. To calculate the accuracy metric, we only focus on the significant pairs, and in the first survey, we discard the uncertain answer "Both headlines seem equally good" for a relatively fair comparison between the two surveys. "Accuracy" measure here is consistent with the LLM tests. We treat the headline with a higher CTR as the correct answer and calculate the proportion of correct answers as accuracy. By doing so, one should expect 50% accuracy by random guess since

Detailed information and survey results are documented in Table A1. Some students did not complete the survey or skipped some comparisons, resulting in 3,809 responses out of a potential 4,040 (202 × 20). To calculate the accuracy metric, we focused solely on the significant pairs, and in the first survey, we excluded the uncertain answer "Both headlines seem equally good" to facilitate a fair comparison between the two surveys. Accuracy here aligns with the LLM tests. We regarded the headline with the higher CTR as the correct answer and calculated the proportion of correct responses as accuracy. As such, we should expect a 50% accuracy rate from random guessing.

The survey results indicate that respondents' ability to identify catchier headlines is comparable to random guessing – the accuracy rate in both surveys is not significantly different from 50% at the 0.05 significance level. Furthermore, accuracies did not vary significantly across different groups based on news reading frequency. These findings suggest that the human ability to discern catchier headlines is limited, highlighting the intrinsic difficulty of the task.

		Survey 1	Survey 2
conducted time		4/14 - 4/21	4/29 - 5/17
# of respondents		202	339
headline pair pool		3000 significant + 500 insignificant pairs	3000 significant pairs
# of sampled pairs in survey		20	20
# total received responses		3,809	4,571
overall accuracy %		45.22	51.58
	Results based on different answer to “How often do you read the news?”		
Never	proportion %	21.78	16.22
	accuracy %	45.76	51.62
1-2 days per week	proportion %	48.51	49.56
	accuracy %	45.36	52.54
3-4 days per week	proportion %	18.81	18.29
	accuracy %	46.83	51.39
5-6 days per week	proportion %	4.95	5.31
	accuracy %	39.05	50.15
Everyday	proportion %	5.94	8.26
	accuracy %	42.35	47.48

Table A1: Survey Response Analysis. No accuracies in this table are significantly lower or higher than 50% at 0.05 significant level. 8 respondents did not finish Survey 2 in the limited given time, so the summation of proportion from different subgroups is smaller than 100% in Survey 2.

B Appendix for Prompt-based Approaches

B.1 Pairwise Comparison Between GPT Prompts

To determine whether one prompt is significantly better than the other, we record a binary indicator of the correct/incorrect answer for each prompt on each sample and then conduct a paired t-test on 1,000 correct/incorrect answer indicators to see whether the answers from one prompt are significantly more accurate than the other. The mean differences and p values are documented in Figure A1 and Figure A2. The random guess model simply predicts the first headline wins with a half probability. The nomenclature for GPT prompts in these two figures follows a specific format. The first component is the name of the GPT model, the second component indicates the number of demonstrations, and the third component specifies whether the outputs in the demonstrations are flipped. A value of 0 indicates the correct label, while a value of 1 indicates the wrong label. For example, GPT-4-0-0 means zero-shot prompting using GPT-4, and GPT-3.5-5-1 means in-context learning with 5 demonstrations and wrong labels using GPT-3.5.

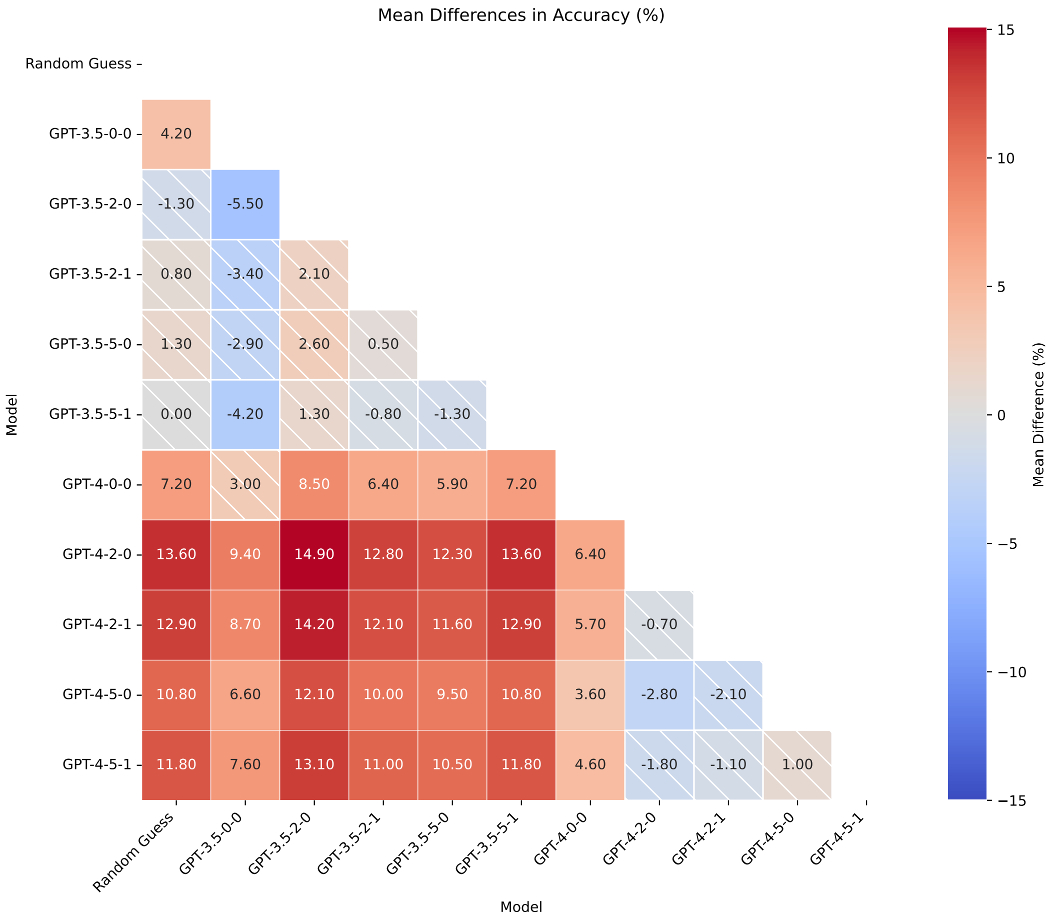


Figure A1: Average accuracy differences between different GPT prompts. The number in the heatmap presents the accuracy of prompts indicated on the left minus the accuracy of prompts indicated at the bottom. Shaded blocks indicate insignificant differences.

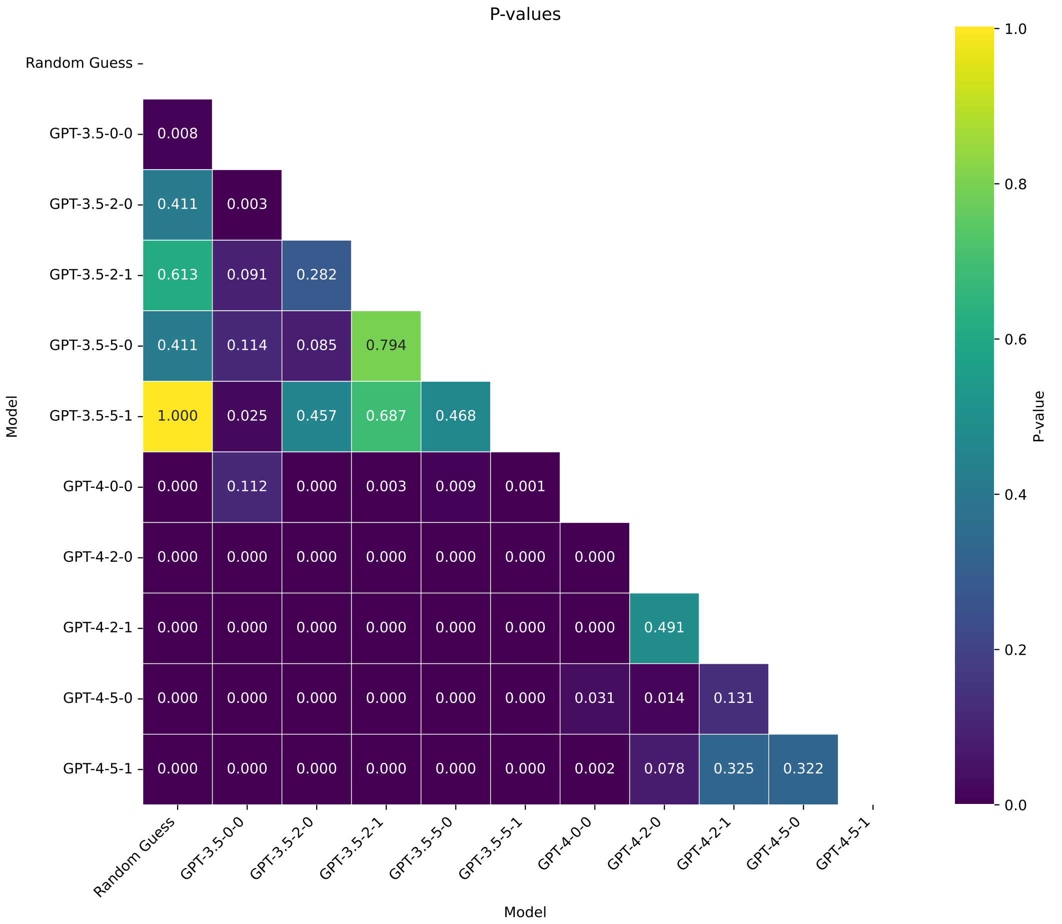


Figure A2: The number in the heatmap presents the p-value from the paired t-test between different GPT prompts.

C Appendix to Embedding-based Classification Approaches

C.1 Implementation Details of MLP Classification Models

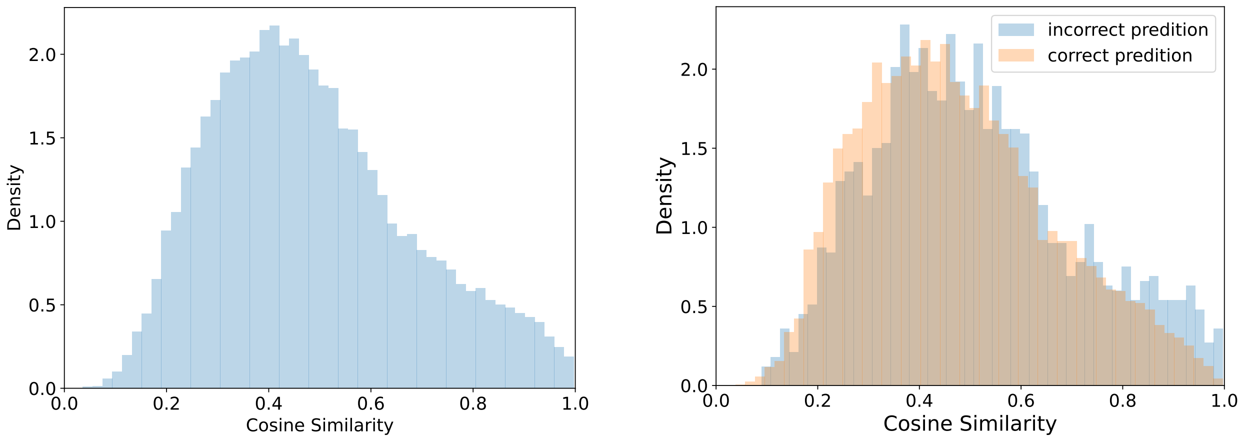
Hyperparameters of the network structure. We employ a fully connected neural network with ReLU activation functions for the hidden layers and a sigmoid activation function for the output layer. This configuration is widely used in practice. To determine the number of layers and the hidden nodes of each layer, we choose the number of layers within the range of $\{1, 2, 3, 4, 5\}$, and set the number of nodes in the first fully connected layer from the set $\{128, 256, 512, 1024, 2048\}$. Besides, we adjust the previous-post ratio, which is the proportion of nodes in each layer relative to those in the subsequent layer. This ratio here is in $\{2, 4, 8\}$. Note that the previous-post ratio becomes irrelevant to the results when there's only one hidden layer. This is because the final output layer has only one dimension for ranking the most attractive headline. We incorporate the dropout after each hidden layer to mitigate the over-fitting. Dropout layers randomly deactivate neurons based on a specified dropout rate, while scaling up the remaining neurons to maintain the same mean of the layer. In this work, we set the dropout rate in $\{0.3, 0.5\}$. All hyperparameters are automatically selected through hyperparameter optimization, which is introduced below.

Hyperparameters of the training procedure. The MLP is implemented in PyTorch, with data split into 70% for training, 10% for validation, and 20% for testing. To prevent the network from memorizing the order of samples, we shuffle the training set at the start of each epoch. Determining the appropriate batch size is crucial, as large batch sizes may lead to convergence to local optima, while small batch sizes may hinder convergence. We balance accuracy and training speed by setting the batch size to 10 and the learning rate to 0.01. Additionally, to mitigate over-fitting, we apply L^2 regularization, with the regularization weight chosen from $\{1e-6, 1e-4, 1e-2\}$. We use binary cross-entropy as the loss function and employ stochastic gradient descent for training. We implement early stopping by monitoring the validation set loss, halting the training process if the reduction in loss over 60 epochs falls below a tolerance threshold of 10^{-4}.

Hyperparameter optimization. We optimize the parameters mentioned above through Bayesian Optimization (Snoek et al., 2012). This Bayesian technique uses Bayesian inference to construct a probabilistic model of the objective function, where the parameters are updated iteratively with new observations. This iterative process balances exploration of the parameter space to discover promising regions and exploitation of known high-performing areas. In this work, we consider the accuracy of the validation set as the objective and optimize hyperparameters selected from the specified parameter space, as delineated by the ranges mentioned previously. We choose the Gaussian process as the estimator with the automatic optimizer, which is commonly adopted by the community. To ensure a judicious trade-off between efficacy and efficiency, we repeat the optimization process 50 times for input data of 256 dimensions and 30 times for data of 3,072 dimensions.

C.2 Effect of Headline Similarity on Performance

We now examine the conditions under which these approaches fail. , Specifically, we investigate whether headline similarity affects the performance of embedding-based classification models. We first calculate the cosine similarity based on the text embeddings for each pair of headlines. This metric captures the similarity between two vectors in inner product space and is defined as the dot product of the vectors divided by the product of their lengths, i.e., $e_1 \cdot e_2/(\|e_1\|\|e_2\|)$, where $e_1 \cdot e_2$ is the dot product of the embedding vectors, and $\|e_1\|$ and $\|e_2\|$ are the their Euclidean norms. Figure A3a shows the distribution of cosine similarity across all headline pairs, whereas Figure A3b shows the distributions of cosine similarity separately for two cases: (1) cases where our embedding-based classification correctly predicts which headline is catchier, and (2) cases where our embedding-based classification incorrectly predicts which headline is catchier. We observe a rightward shift in the distribution of incorrect predictions (compared to the distribution of correct



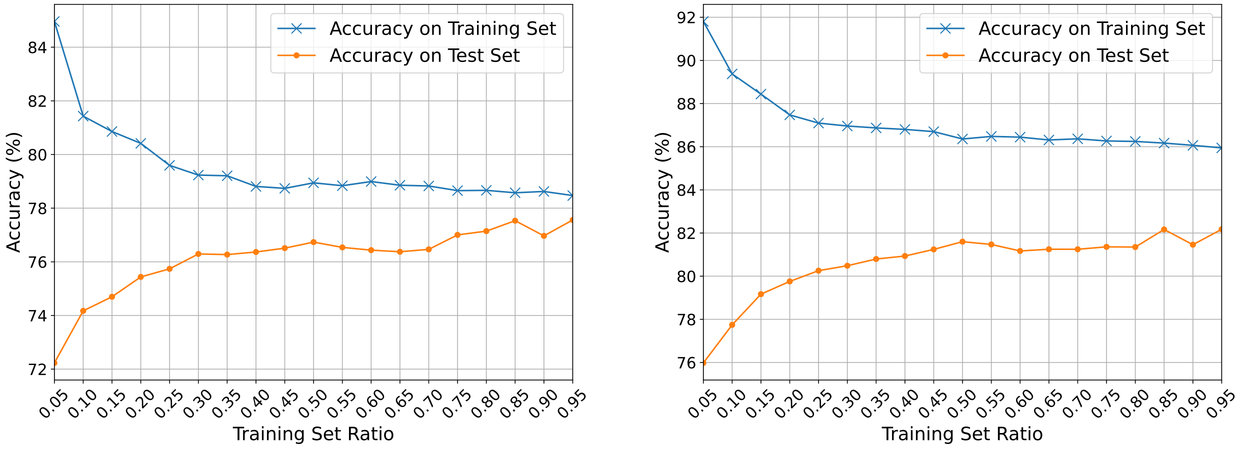
(a) Cosine similarity distribution over all pairs in training and test sets

(b) Cosine similarity over incorrect/correct predicted test pairs

Figure A3: The left panel shows the distribution of cosine similarity for all pairs. The right panel shows the distributions of cosine similarity for both incorrect and correct predictions using the logit classification model with OpenAI-3072E embeddings.

predictions). This suggests that incorrect predictions are more common when the headlines are very similar to each other (i.e., higher cosine similarity). There are two possible reasons for this. First, in cases where the headline pairs are very similar, there is a higher chance of Type 1 error (in the t-tests based on our A/B test data), i.e., the headlines may not be significantly different in fact. Second, text embeddings may not be able to capture the differences between two headlines when they are similar, i.e., the embeddings may also be very similar. As a result, it is hard for the classification model to differentiate between them.[19]

C.3 Effect of Training and Test Data Size on Performance



(a) Performance of OpenAI-256E

(b) Performance of OpenAI-3072E

Figure A4: Performance of Logit classification models with OpenAI embeddings under varying training dataset sizes and different embedding sizes.

We now conduct experiments varying the proportion of data used for training vs. test. We use the

[19] Note that it is unlikely that this phenomenon is due to problems/issues with the classification model because both Logit and MLP achieve nearly identical performance.

Logit model for these experiments since it offers the best performance and is easy to train. The results from this exercise are shown in Figure A4. As we can see, when we use the 256-dimensional embeddings, approximately 65% of the training set, comprising approximately 25,000 pairs (with the remaining 35% used for the test set), is adequate to achieve high and stable performance. However, when we use the 3072E-dimensional embedding, there is a clear over-fitting[20] issue even if we use the simplest Logit model and 95% of the data for training. This suggests that for larger embedding vectors, we need much larger training datasets.[21]

C.4 Dimensionality Reduction and Regularization

In this section, we examine whether employing dimensionality reduction techniques and/or regularization can improve performance and reduce overfitting.

C.4.1 PCA for Embeddings

One can observe that OpenAI-256E in Figure A4(a) does not suffer from the over-fitting in contrast to OpenAI-3072E as shown in Figure A4(b). Thus, one straightforward method one can try to attack the over-fitting issue is reducing the dimension of embeddings using PCA or directly using lower-dimensional OpenAI embeddings. However, as we have discussed that OpenAI-256E has a worse performance than OpenAI-3072E regarding the accuracy in the testing dataset, we turn to PCA to see if PCA-reduced embeddings can improve the performance by solving the over-fitting issue.

Specifically, we first use PCA to reduce 3072-dimensional OpenAI embeddings. Then, we concatenate two reduced embeddings from healing pairs and use Logit for the downstream classification task. The result is displayed in Figure A5, which shows that as the dimension is reduced, although there is a smaller accuracy gap between the training and test set, the accuracy on both datasets decreases, indicating that PCA does not work for our task, which prefers higher testing accuracy.

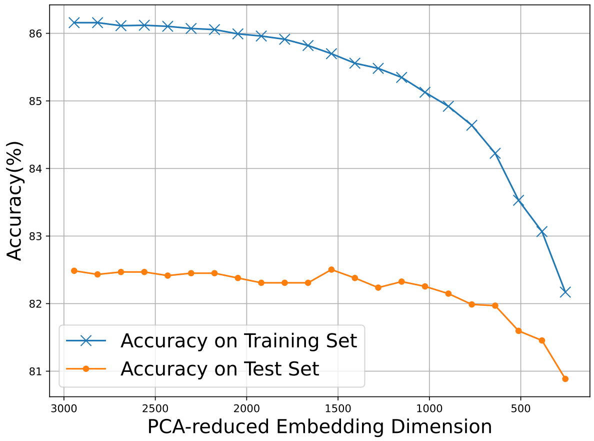


Figure A5: Accuracy of the Logit classification model with varying PCA-reduced OpenAI embeddings as inputs.

[20] Over-fitting refers to the phenomenon that the accuracy on the training set is consistently higher than the accuracy on the test set. It occurs when a machine learning model learns the details and noise in the training data to the extent that it negatively impacts the model's performance on new, unseen data. This often happens because the model becomes too complex or because the training set is too small, causing the model to memorize specific examples and noise rather than general patterns.

[21] In principle, overfitting can also be due to model complexity. However, this is unlikely to be the case in our setting since we are using a simple Logit model with linear features.

C.4.2 L^1, L^2 Regularization

Another widely applied method to deal with over-fitting is adding regularization terms in the loss function. This was done using the `penalty` (indicating L^1 or L^2) and `C` (indicating the inverse of penalty weights, and the default value is 1) control parameters in the `LogisticRegression` function from `sklearn` package. We tested both L^1 and L^2 regularization with different weights, i.e., $1/\texttt{C} \in \{0.05, 0.1, 0.15, 0.2, \ldots 2\}$. The results are shown in Figure A6; the accuracy gap diminishes as the regularization weight increases, but the accuracy first increase and then decrease on the test set (the accuracy continues to decrease on the training set). This result suggests that one can just the default L^2 regularization with $\texttt{C} = 1$ provided in the `sklearn` package to get almost the best performance due to the proper weighted regularization term. However, similar to PCA, although regularization largely closes the gap due to over-fitting, it decreases the accuracy at large when the gap is small.

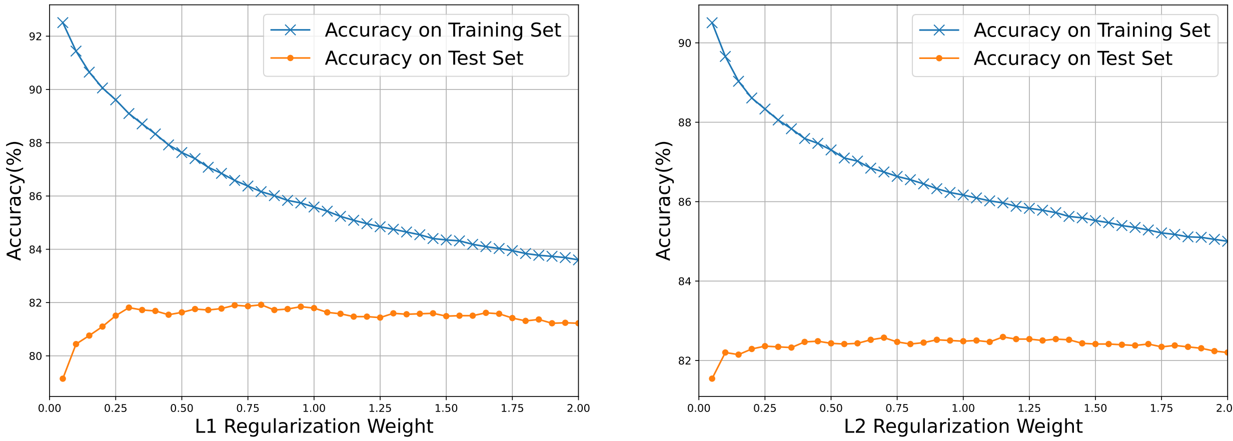


Figure A6: Accuracy of the Logit classification model with varying L^1 (left) and L^2 (right) regularization weights. X-axis is the regularization weight defined as $1/\texttt{C}$.

In sum, we find that neither PCA nor regularization improves the performance accuracy on the test data, which suggests that over-fitting is unlikely to be due to learning noise in the training data. Our analysis suggests that we need more A/B tests to learn the relationship between a headline's text embeddings and its attractiveness effectively. In summary, these comparisons suggest that data samples from Upworthy are insufficient to unlock the potential of LLMs fully. Nevertheless, diverting more and more traffic to A/B tests is costly, and this observation is one of the motivations for our approach later in §4.

C.5 Potential Threats and Robustness Checks

Next, we discuss two potential threats to the inference from our analysis here and our robustness checks to alleviate concerns around these threats. First, one may be concerned as to whether Upworthy's data has been used as part of OpenAI's training corpus – which could potentially invalidate our analysis. However, this is unlikely for several reasons. First, it is improbable that OpenAI would have specifically downloaded these Upworthy CSV files (Comma-Separated Values files) dataset from a subfolder in Upworthy Research Archive. Second, the original data only contains impressions and clicks for each headline, as shown in Table 1, rather than paired headlines with winner information. Therefore, during the training of OpenAI models, it is unlikely they were exposed to the winner information since the training objective is to predict the next word. Third, we conducted an experiment to determine if this dataset was indeed a part of OpenAI's corpus. We used embeddings from two headlines with significantly different impressions (possibly from different tests) for the downstream task of predicting which headline received higher impressions. If the embedding model was trained on this corpus, its embeddings should reflect the number of impressions and perform well on this task. However, the observed accuracy was approximately 50%, indicating that the embeddings do not contain such information, confirming our assumption. Nevertheless, if a firm is concerned that its data is a part of public repositories or is concerned about privacy issues around submitting headlines to OpenAI to obtain embeddings, then instead of using embedding-based approaches, it can rely on fine-tuning based approaches; see §3.3 for details.

Second, one may be concerned that readers' preferences changed over time and that the current sample splitting method fails to take into account when the headline was created, making our classification task invalid. To address this concern, we conducted a robustness check by splitting the dataset based on the creation time of the headlines (e.g., the initial 80% of the data, in chronological order, as the training set, and the subsequent 20% as the test set). We found its accuracy was 81.50% (close to the best embedding accuracy of 82.50%), indicating that time shift preference is not a concern in our data.

D Appendix to LoRA Fine-Tuning

D.1 Transformer and Llama Architecture

The Transformer model, introduced by Vaswani et al. (2017), has revolutionized the field of NLP with its innovative attention mechanism and highly parallelizable architecture. Unlike traditional recurrent neural network (RNN) based models, the Transformer uses self-attention mechanisms to process input data. This shift allows for greater computational efficiency and significantly improved performance across various NLP tasks.

As illustrated in the right portion of Figure A7, the Transformer block comprises multiple layers, each featuring two primary components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Key enhancements include layer normalization (Ba et al., 2016) and the use of residual connections (He et al., 2016), which improve training stability and help prevent issues such as vanishing gradients. While some specifics are omitted to maintain brevity and readability, it's important to note that such architectures are subject to rapid evolution and vary across different LLMs.

Self-Attention Mechanism: The most important innovation in the Transformer block is the self-attention mechanism. It enables each token in a sentence to "pay attention" to all other words when computing its own representation. This is achieved using an attention mask that guides which tokens should influence each other, thereby allowing the model to capture dependencies between words regardless of their distance in the text. Specifically, for a given input sentence, the model first converts each token into an embedding,[22] forming a matrix X where each row corresponds to the embedding of a token. If the sentence contains n tokens, and each token is represented by a d-dimensional embedding, then X is an $n \times d$ matrix. From these embeddings, the model generates three matrices: Query Q, Key K, and Value V:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where W_Q, W_K, and W_V are trainable weight matrices of dimensions $d \times d_q$, $d \times d_k$, and $d \times d_v$ respectively. d_q and d_k are typically set to be the same, allowing for the dot-product operation used in computing the attention scores to be valid.

The attention scores are computed by first taking the dot product of the query matrix Q and key matrix

[22] This embedding differs from the text embedding used in the previous section. Each embedding here simply corresponds to one token without paying attention to the whole input sequence. The text embeddings used in the previous section are essentially the contextualized embeddings, which are generated by passing the tokens through multiple layers of Transformers, capturing rich contextual information, and summarizing the whole input sequence.

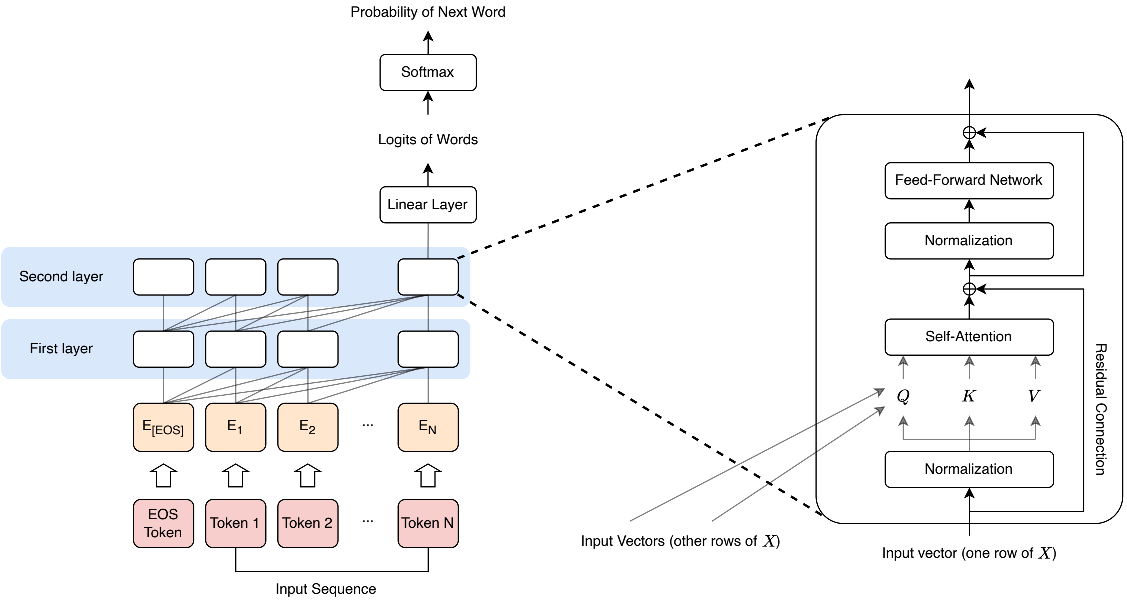


Figure A7: Illustration of the Transformer block and architecture typical in most LLMs.
The left portion of the figure displays the overall architecture of an LLM, highlighting the layers of Transformer blocks and the connections that represent the attention mask. Lines between Transformer blocks in different layers indicate where attention is applied; the absence of a line signifies no attention interaction. Specifically, a causal attention mask is depicted, ensuring that predictions for each token depend only on preceding tokens. Note that this figure represents one type of attention mask; other types are also employed in various LLMs but are not depicted here. Additionally, only two layers are shown for simplicity; actual models may contain many more layers and connections, which are not depicted here to maintain clarity and focus on key components.
The right portion provides a detailed view of a single Transformer block, including internal components. We omit the depiction of multi-head attention for brevity. Note that different LLMs may use slightly different designs.

K, then scaling by the square root of the key vector dimension d_k, and applying an attention mask M. A softmax function is applied subsequently:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

The attention mask M typically includes negative infinity at positions where attention is not applicable, ensuring the attention weights are effectively zero after applying the softmax. At positions where attention is relevant, the mask values are zero. In models like GPT and Llama, a causal attention mask, a.k.a, autoregressive attention, is utilized to prevent future tokens from influencing the prediction of the current token during training. The causal mask is a lower triangular matrix, where positions above the diagonal are set to negative infinity, and positions on and below the diagonal are set to zero. While BERT employs a bidirectional attention mechanism that allows each token to depend on all other tokens in the sequence.

Multi-Head Attention: To capture diverse aspects of the relationships between tokens, the Transformer enhances the self-attention mechanism through the use of multi-head attention. This approach involves performing multiple self-attention operations in parallel, with each "head" having its own independent Q, K, and V matrices. The outputs from these operations are concatenated and then linearly transformed to produce the final attention output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O,$$

where each head computes an independent self-attention process:

$$\text{head}_i = \text{Attention}(XW_Q^i, XW_K^i, XW_V^i),$$

with W_Q^i, W_K^i, and W_V^i being the learnable parameter matrices specific to the i-th head. W_O is the weight matrix with dimension $hd_v \times d$, and is applied to the concatenated outputs from all heads, integrating the distinct perspectives captured by each head into a unified output.

Feed-Forward Networks: Each layer of the Transformer also includes a position-wise fully connected feed-forward network. This network comprises two linear transformations with a ReLU activation function in between:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where W_1, W_2, b_1, and b_2 are learnable parameters. The final output of the FNN has the same dimension d as the input embeddings. This network acts on each position separately and identically, providing additional non-linearity and enhancing the model's ability to learn complex patterns in the data.

Llama Architecture: The Llama model enhances the basic Transformer architecture to handle vast amounts of data and leverage extensive computational resources effectively, establishing it as one of the most powerful open-source language models currently available. Llama-3 retains essential components such as multi-head self-attention and feed-forward layers from the original Transformer model but incorporates several techniques to stabilize training and boost efficiency and performance. These enhancements include the application of pre-normalization with RMSNorm (Zhang and Sennrich, 2019), the SwiGLU activation function (Shazeer, 2020), rotary positional embeddings (Su et al., 2024), and grouped-query attention (Ainslie et al., 2023), among others. Like Llama-3, other LLMs share a similar Transformer architecture but vary in the number of Transformer layers, training corpora, size of the weight matrices, activation functions, etc.

In the multi-layer structure of LLMs, the output of one layer serves as the input to the subsequent layer. Each layer's output is a matrix of the same dimensions as the input matrix X, with each row representing the transformed embedding of a token, enriched with context from other tokens through the self-attention mechanism. After processing through all the layers, the final output matrix contains the contextualized embeddings of all tokens in the input sentence. Unlike RNNs that process input sequences one token at a time, the Transformer architecture can process whole input sequences in parallel, significantly enhancing processing speed and efficiency.

It is worth noting that LLMs do not necessarily operate in a conversational manner, nor do they always use language as their output. As we will demonstrate in §3.3.2, we employ the Llama model as a classifier, where the output is the distribution of predicted classes. In applications like conversational AI, exemplified by GPT, the contextualized embeddings from the last token of the last layer of the Transformer are fed into a linear layer followed by a softmax function, the configuration of which transforms these embeddings into a probability distribution over the vocabulary, enabling the model to predict the next token in the sequence.

D.2 Additional LoRA Models

D.2.1 Incorporating Insignificant Pairs for LoRA Fine-tuning

Similar to the embedding-based logit model, we have experimented with various LoRA configurations, such as further reducing the number of trainable parameters, increasing the dropout ratio, and applying larger weight decay in an attempt to mitigate over-fitting. However, as we observed, these modifications generally reduce performance on the training set without yielding significant improvements on the test set.

In the following, we test whether incorporating more data, including insignificant pairs, can help improve the performance. Notice that by incorporating more significant pairs, we indeed observed improvement in test accuracy (see Figure A4), the observation of which holds for both embedding-based models and LoRA fine-tuned models.

Here, we only present the results from fine-tuning Llama-3-8b using different sizes of training data, and the result from the embedding-based model remains the same. Recall that in §3.3.3, we use only significant pairs (with p-values smaller than or equal to 0.05, which includes 39,158 pairs) and split them into training and test sets. We continue to use the same significant test set for performance evaluation. However, for the training set, we obtain the data in the following manner: first, we remove the fixed test set from all pairs, leaving a total of 134,941 pairs. Then, given a ratio (e.g., 60%), we select these remaining pairs' first 60% (ranked by ascending p-value) as the training set, see Figure A8 for a diagram of our dataset construction.

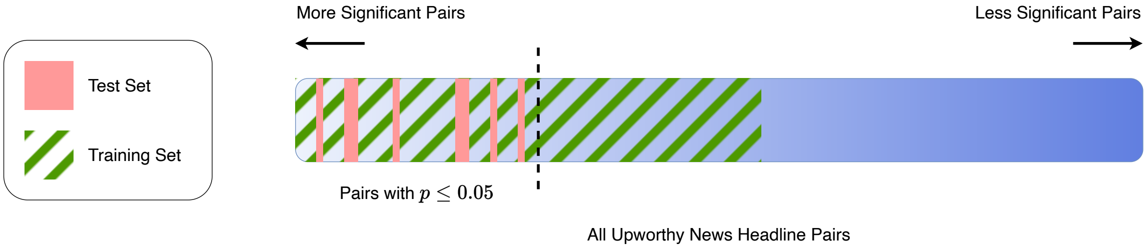


Figure A8: Dataset Construction by Incorporating Both Significant and Insignificant Pairs in Llama-3-8b LoRA Fine-tuning

We set the ratio to 20%, 40%, 60%, 80%, and 100%. For each ratio and size of the training set, we finetune the Llama-3-8b model and evaluate the performance on the fixed test set. Except for the training set, all other training configurations are the same as in §3.3.2. The performance is reported in Figure A9. The accuracy of the training set decreases as the ratio gets larger since we are incorporating more insignificant pairs. The accuracy on the test set maintains around 83%, which indicates that incorporating more training data does not necessarily lead to better performance, at least for the current dataset. Notice that the decreasing trend of accuracy on the training set in Figure A4 is less steep because the pairs in that training set are all significant. However, the main goal here is to confirm that more insignificant pairs cannot improve the performance for either LoRA or the embedding method.

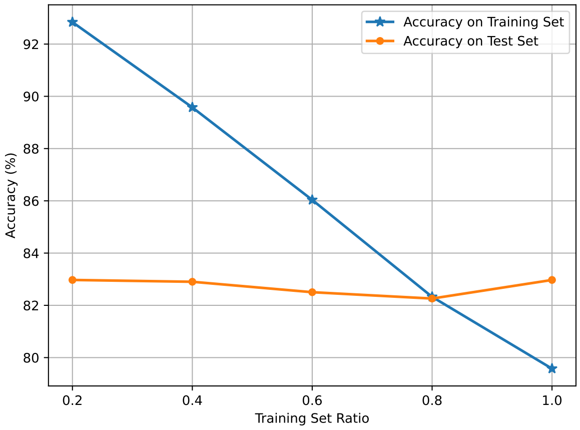


Figure A9: Performance of fine-tuning Llama-3-8b under varying training dataset sizes.

D.2.2 DistilBERT LoRA Fine-Tuning

We now discuss One difference between DistilBERT and Llama-3 is that DistilBERT is a bidirectional Transformer model, which means that its attention mechanism allows the model to learn the context from both directions, while Llama-3 is a unidirectional model. An illustration of the structure of the DistilBERT model combined with a classification head is shown in Figure A10. By default, DistilBERT can handle two headlines together as input. The format of the input is: `[CLS] Headline 1 [EOS] Headline 2 [EOS]`. The `[CLS]` token is a special token used to get the output of the model for classification, and the output of the last Transformer layer corresponding to this token is used as the input to the classification head. The `[EOS]` token is used to separate the two sentences.

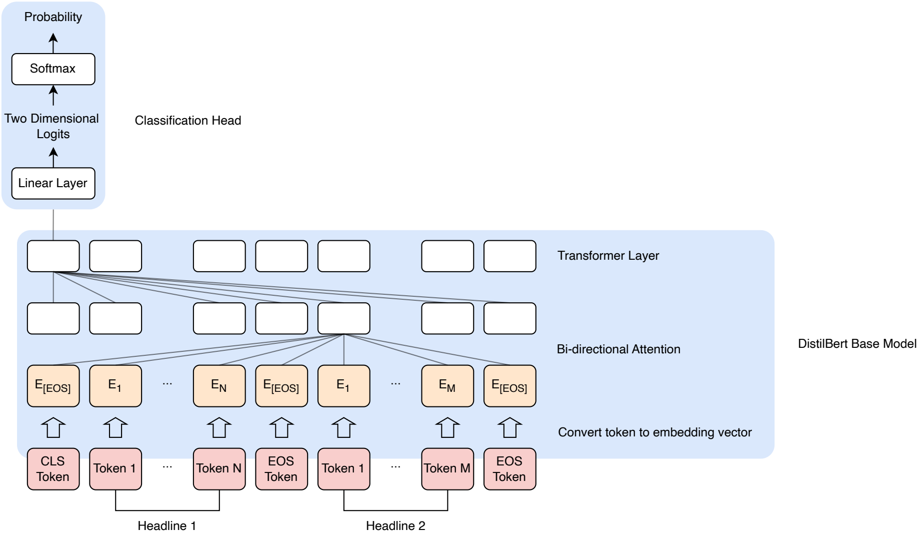


Figure A10: The classification pipeline using the DistilBERT model. Headlines are first tokenized, and each token is converted into an embedding vector. For illustrative purposes, only two Transformer layers are shown. The figure highlights the use of bidirectional attention in DistilBERT, whereby each token can incorporate information from any token. Connections between Transformer blocks represent the attention mechanism, and only a subset of these links are displayed for simplicity. The output from the first token (CLS token) is fed into a classification head, which computes the probability that one headline is more impactful than the other.

Another difference is that DistilBERT is a smaller model compared to Llama-3-8b. DistilBERT has around 66 million parameters, which is only 0.94% of the parameters of Llama-3-8b. Due to the smaller model size, we are able to conduct full parameter updates during the fine-tuning process without using LoRA. We use the same learning rate scheduling, number of training epochs, and train-test set split as in the experiment detailed in §3.3.2. The accuracy of DistilBERT LoRA fine-tuning on the same test dataset is around 75%. Thus, we find that the newer Llama-3-8b model offers better performance upon fine-tuning. We refer interested readers to our code repository for further details on the fine-tuning process.

E Cost Comparison of Pure-LLM-Based Methods

In Table A2, we summarize the monetary cost and effort required for each pure-LLM method used in this paper.

For the prompt-based method utilizing OpenAI API, the cost was $1.13 for 5,000 GPT-3.5 prompts and $20.11 for 5,000 GPT-4 prompts, which is approximately 20 times the cost of GPT-3.5. The total run-time for the entire experiment was 121.5 minutes, involving the testing of 10,000 prompts.

Method	Cost	Runing time	Effort
Prompt	$21.24 (10,000 GPT prompts)	121.5 mins for 10,000 GPT prompts (82 prompt requests per min)	Low
Embedding	$0.18 (`text-embedding-3-large`)	26 mins for embedding all 77,245 headlines (3,000 embedding requests per min)	Moderate
LoRA	$3 (Llama-3-8b)	2 hours (Llama-3-8b)	High

Table A2: Summary of cost, run-time, and effort for different methods. Note that the reported run-time for Prompt-based (GPT-3.5 and GPT-4) and embedding-based methods refer to the time needed to obtain responses or embeddings from OpenAI. Note that the prompt and embedding response time can vary based on the API usage tiers, with our results specifically derived from Tier 1.

For the embedding-based methods, obtaining text embeddings via the OpenAI API is extremely time-efficient and more economical compared to prompt-based methods. When using the embedding API, data transmission involves smaller, fixed-size vectors instead of variable-length text, which reduces data overhead and accelerates processing speed. We spent approximately $0.18 to obtain text embeddings for all 77,245 headlines, while the prompt experiment with only 10,000 prompts cost around $20.[23] Additionally, processing embeddings is much faster (the maximum Tier 1 rate limit—3,000 requests per minute reached) compared to prompt-based methods (around 82 pairs per minute). This efficiency can lead to lower operational costs and easier integration into real-time applications.

For the LoRA fine-tuning method, the GPU requirements depend on factors such as model size, batch size, optimization algorithm, and floating-point precision. In our experiment, full-parameter fine-tuning without LoRA would require over 100 GB of GPU memory. However, LoRA fine-tuning only requires 30 GB of GPU memory. To illustrate the benefit of reduced GPU requirements, we compare two commonly used GPU models for LLM training. Without LoRA, the NVIDIA H100 GPU, which has nearly the largest GPU memory (80 GB), is insufficient for fine-tuning. In contrast, with LoRA, a more affordable NVIDIA V100 GPU with 32 GB of memory is sufficient. The price of an NVIDIA H100 GPU is ten times higher than an NVIDIA V100 GPU as of May 2024. Thus, using LoRA significantly reduces costs, making fine-tuning more accessible to researchers and practitioners. In our experiment, each replicate involved approximately 2 hours of fine-tuning the Llama-3-8b model on an NVIDIA A100 GPU with 80 GB memory, costing about $3 on the cloud computing provider AutoDL. We also note that after fine-tuning the model, several cost-efficient techniques are available for deploying it for inference, significantly reducing computational costs and hardware requirements. For instance, `llama.cpp` allows LLM models to be deployed on machines without a GPU, making implementation more affordable. Moreover, model compression techniques such as quantization, sparsification, and knowledge distillation can drastically cut inference computation costs while maintaining, or only slightly affecting, model performance. For a comprehensive survey on efficient LLM deployment and inference techniques, please refer to Zhou et al. (2024).

In summary, we recommend using either embedding-based methods or fine-tuning methods, as they achieve similar performance (at least in our setting), superior to prompt-based methods, and are cost-effective. These methods offer different levels of transparency: prompt-based methods using OpenAI's API are entirely black-box; embedding-based methods use OpenAI's black-box embedding model combined with a user-controlled classification model; while fine-t tuning open-source models provides full transparency.

[23] At the time of this study (May 2024), GPT-4 (turbo version) prompt costs $10 per 1M input tokens and $30 per 1M output tokens. The embeddings used in our study, `text-embedding-3-large`, only cost $0.13 per 1M input tokens and there is no output token cost associated with embeddings. Our entire dataset comprises 77,245 headlines, with an average of 19 tokens per headline, resulting in a total embedding cost of approximately $0.18.

Fine-tuning methods require more significant engineering effort to fine-tune and deploy an LLM than embedding-based methods.

F LLM-based CTR Prediction Models

We compare two approaches for CTR prediction: an OpenAI-embedding-based model and a LoRA fine-tuned Llama-3-8b model. In summary, we find the LoRA approach has better performance and we finally choose this model in the first step of our LOLA algorithm.

The OpenAI-embedding-based CTR prediction model is similar to the one used for binary classification, as shown in Figure 3, with three key modifications: changing the loss function from log loss to MSE, using the text embedding of a single headline (3072 dimensions) instead of the concatenated embedding from headline pairs, and switching from binary output to continuous CTR prediction.

To fine-tune the Llama-3-8b model for predicting CTR, we use a slightly different model structure compared to the one used for the pairwise comparison task shown in Figure 5. Specifically, as illustrated in Figure A11, we still use the Llama-3-8b model as the base LLM. The input is a single headline, and we apply a linear transformation to the output vector of the final token from the last Transformer layer. This transformation produces a single value, representing the predicted CTR. The model is trained to minimize the MSE between the real CTR and the predicted CTR. All other hyperparameters, such as batch size, number of training epochs, learning rate and its scheduling, and LoRA-related parameters, remain unchanged as in §3.3.

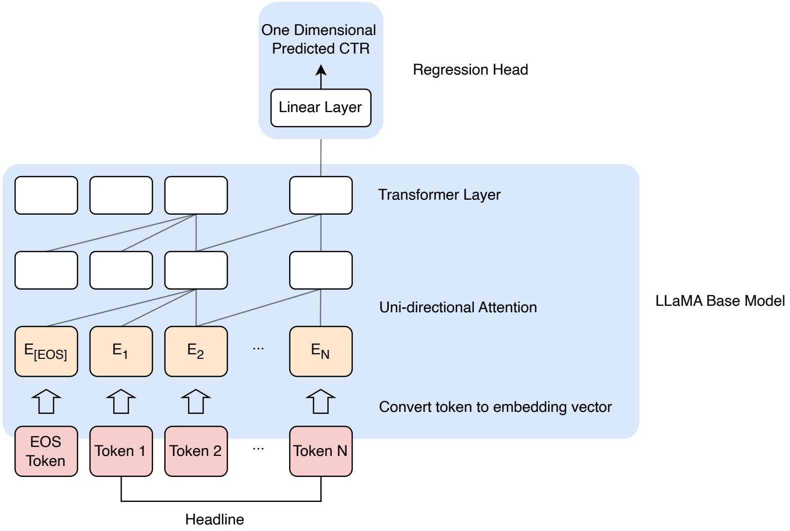


Figure A11: Model architecture for CTR prediction based on the Llama-3-8b model.

Figure A12 presents the numerical results, showing both the training loss (MSE) and accuracy throughout the training process. Accuracy is computed as follows: for each dataset (training or test), we use the fine-tuned model to predict the CTR for each headline. For each news article, if the headline with the highest predicted CTR matches the headline with the highest actual CTR, we consider this prediction correct. The accuracy is then calculated as the proportion of news articles for which the model's predictions are correct.

As shown in Figure A12, the fine-tuned Llama-3-8b model significantly outperforms the OpenAI embedding-based method in terms of accuracy. Regarding the R-squared values, the embedding-based approach achieves 0.241, whereas the LoRA fine-tuned method achieves a higher R-squared value of

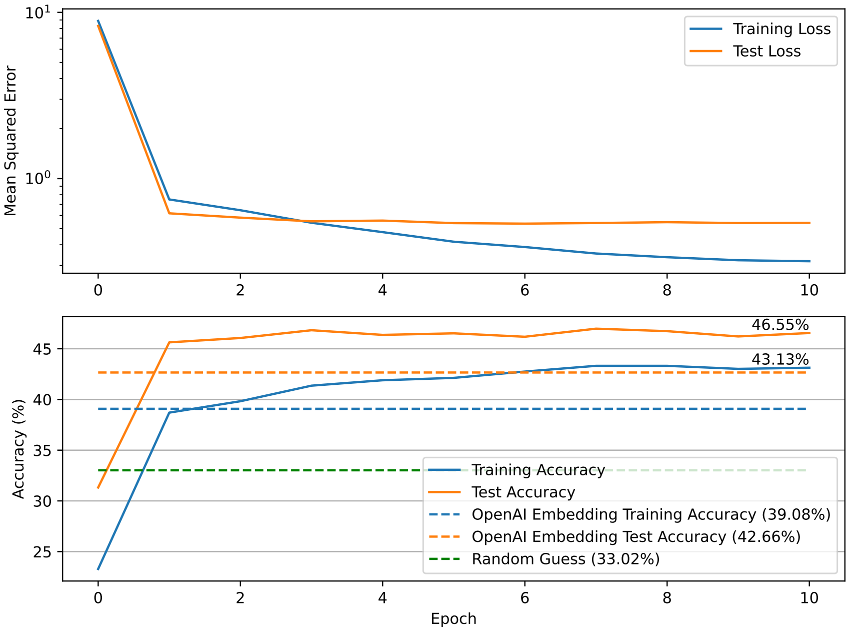


Figure A12: Performance of fine-tuning the Llama-3-8b model for CTR prediction. The accuracy of the OpenAI embedding-based method and the random guess method are also reported. For the random guess method, the winner headline is uniformly and randomly selected for each news article.

0.315. In contrast, for the pairwise comparison task, the fine-tuned Llama-3-8b model shows only a marginal improvement over the OpenAI embedding-based method. CTR prediction on all headlines is more challenging than pairwise comparison on significant pairs because the CTR output has higher uncertainty compared to a binary label. We believe this increased difficulty necessitates a more complex model, such as Llama-3-8b, which results in a larger performance gap between the fine-tuned Llama-3-8b and the OpenAI embedding-based method.

Additionally, note that the accuracy on the test set is higher than on the training set. In our experiment, many news articles have headlines with very similar CTRs, making it more difficult to distinguish between them, which lowers the accuracy. This result may be due to having more such news articles in the training set and fewer in the test set. It is possible that with a different random split of the training and test sets, the accuracy of the training set could be higher than that of the test set.

G Details of LOLA Variants and Extensions

G.1 LOLA with Best Arm Identification

In this section, we showcase how the LOLA framework can be used to solve the best arm identification problem.

The best arm identification problem is a variant of the multi-armed bandit problem, where the goal is to identify the arm (or arms) with the highest reward. This problem is particularly useful in practice when the objective is to identify the best arm as quickly as possible rather than optimizing cumulative reward. Best arm identification has many variants, and our LOLA framework can be effectively integrated into them. We present the general steps of such a best-arm identification algorithm and the integration of LOLA in Algorithm 2.

Algorithm 2 LLM-Assisted Best Arm Identification (LLM-BAI)

```
LLM Training Phase: Train a LLM-based prediction model $\mathcal{M}(x)$ for CTRs using historical data samples, where $x$ is the contextual information for arms.
Hyperparameter Fine-Tuning: Use another subset of data to choose the LLM's equivalent auxiliary sample size, $n^{\text{aux}}$.
Online Learning Phase: Initialize the number of arms $K$, the number of initial pulls $T_k \leftarrow n^{\text{aux}}$, LLM-based CTR prediction $\bar{\mu}_k \leftarrow \mathcal{M}(x_k)$ for all arms $k \in [K]$, and the candidate set as $\mathcal{C} \leftarrow [K]$.
while $|\mathcal{C}| > 1$ do
    Pull each arm in $\mathcal{C}$ once, denote the observed reward for arm $k$ as $r_k$.
    Update the sample average CTRs $\bar{\mu}_k \leftarrow (\bar{\mu}_k T_k + r_k)/(T_k + 1)$, update $T_k \leftarrow T_k + 1$ for all arms $k \in \mathcal{C}$.
    Update confidence interval for each arm as $\text{CI}_k \leftarrow [\bar{\mu}_k - C(T_k), \bar{\mu}_k + C(T_k)]$, for all arms $k \in \mathcal{C}$, where $C(T_k)$ is a decreasing and non-negative function of $T_k$.
    Eliminate suboptimal arms from $\mathcal{C}$, i.e., eliminate arms from $\mathcal{C}$ if their upper confidence bound is less than the lower confidence bound of some other arms.
end while
Return: The set of good arms.
```

We compare the performance of our proposed LLM-BAI to the standard benchmarks, including the standard A/B test, pure BAI algorithm, and pure LLM. Similar to the numerical setting under the regret minimization framework, we treat the reported CTR in the Upworthy dataset as the true expected reward of each headline. Each bandit problem instance corresponds to one news item, and each news headline is viewed as one arm. The return of each arm follows a Bernoulli distribution with the expectation given as the reported CTR.

We observe that many news items in the Upworthy dataset have headlines with similar CTRs, making the best arm identification problem challenging. In real-world practice, the benefit of choosing the best arm is quite small (or non-existent) when the difference between the best arm and other arms is small (or insignificant). Therefore, we select the $(\text{ST})^2$ algorithm from Mason et al. (2020) as the pure BAI algorithm, which is tailored to scenarios where the difference between the best arm and the suboptimal arms could be small.[24] Specifically, if we denote the real expected return of each arm as μ_k and the expected return of the best arm as $\mu^* = \max_{k \in [K]} \mu_k$, and given thresholds ϵ and γ, and failure rate δ, this algorithm returns a set of arms containing all arms with expected returns better than $\mu^* - \epsilon$, and no arms with expected returns worse than $\mu^* - \epsilon - \gamma$ with high probability. In our case, we set $\epsilon = 0$, $\gamma = 0.005$, and $\delta = 0.2$. It means, with at least 0.8 probability, the algorithm returns a good set, which does not contain any bad arm with CTR lower than $\mu^* - 0.5\%$. We make slight modifications to the original algorithm for numerical stability; for details, please see our code at LLM News Github Repository.

In the implementation, we adopt the same data splitting rule as described in §4.3. Specifically, we used 70% data to fine-tune the Llama-3-8b CTR prediction model (see details in Web Appendix § F), 10% data to fine-tune the n^{aux} parameter, and tested the performance of the LLM-BAI algorithm on the remaining 20% test dataset. To fine-tune the n^{aux} parameter, we run the LLM-BAI on the algorithm tuning dataset with different n^{aux} values in $\{20, 50, 100, 200, 300, 400, 500, 700, 1000\}$, and choose the one with highest success

[24] Note that we can also consider a similar setting in the regret minimization framework, where the planner's goal is to ensure that the regret is below a certain threshold rather than specifying it in comparison to the best arm, as in Equation (1); see Feng et al. (2024) for details.

rate in BAI. We finally choose $n^{\text{aux}} = 300$ as the best parameter since it has the highest success rate and the number of pulls is not significantly more than others.

For the natural benchmark pure LLM, we use the same CTR prediction model as LOLA. We also implement the plain BAI without using LLM-based predicted CTR as priors. For the last benchmark, the A/B test method, we allocate a fixed number of pulls to each headline, pulling all headlines an equal number of times. We then calculate the empirical CTR for each headline and select the one with the highest empirical CTR. To evaluate the performance of the A/B test method, we run experiments using several different numbers of pulls per headline, specifically $\{200, 300, 400, 500, 600\}$.

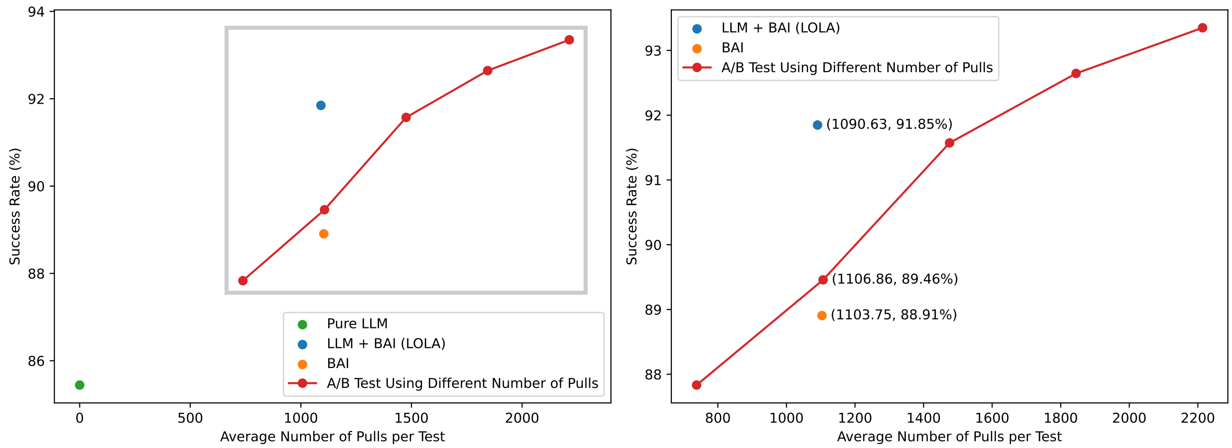


Figure A13: Performance of pure LLM, LOLA, BAI, and A/B test on the test dataset. The X-axis represents the average number of pulls per test (news). The Y-axis represents the success rate. A test is considered successful if all selected headlines (both LOLA and BAI could return multiple good arms) have CTRs above $\mu^* - 0.5\%$.
The right figure zooms in on the area boxed by the gray rectangle in the left figure to highlight the performance improvement of our LOLA method.

We report the performance of LLM-BAI and the standard benchmarks in Figure A13. Note that the success rate for pure LLM here is higher than the accuracy (46.55%) reported in Figure A12, as we use a more lenient definition of success. Our results indicate that our LOLA algorithm outperforms both the pure BAI method and the A/B test method in terms of success rate when using a similar number of pulls per test. Specifically, LOLA improves the success rate by an absolute 2.39% and saves 1.47% of pulls compared to the A/B test. Compared to the plain BAI method, LOLA improves the success rate by an absolute 2.94% and saves 1.19% of pulls.

G.2 LOLA with Thompson Sampling

We present a version of LOLA (LLM-TS) that combines LLM and Thompson Sampling and compare its numerical performance with standard Thompson Sampling and LLM-2UCBs.

LLM-TS extends the standard TS algorithm by initializing an LLM-based prior. See Algorithm 3 for details. Similar to LLM-2UCBs, here also we need to fine-tune the hyperparameter, n_k^{aux} in this case, which is the LLM's equivalent auxiliary sample size. As in the main analysis, we fine-tune this parameter on the fine-tuning dataset and choose from the set In our numerical experiment, we consider the $n_k^{aux} \in \{600, 800, 1000, 1200, 1400\}$ and find that $n_k^{aux} = 1200$ generates the highest clicks in the algorithm tuning dataset (which we then use in all our numerical experiments). Notice that this number is quite close to 1000, which was the number used in LLM-2UCBs. There is no significant difference using either 1000

Algorithm 3 LLM-Assisted Thompson Sampling (LLM-TS)

```
**LLM Training Phase:** Train a LLM-based prediction model $\mathcal{M}(x)$ for CTRs using historical data samples, where $x$ is the contextual information for arms.
**Hyperparameter Fine-Tuning:** Use another subset of data to select the LLM's equivalent auxiliary sample size, $n^{aux}$ and the standard TS prior $(\alpha^0, \beta^0)$.
**Online Learning Phase:** Initialize the number of periods $T$, number of arms $K$, LLM-based CTR prediciton $\bar{\mu}_k^{aux} \leftarrow \mathcal{M}(x_k)$, the parameters in Beta priors $(\alpha_k^1, \beta_k^1) \leftarrow (\alpha^0, \beta^0) + (n^{aux}\bar{\mu}_k^{aux}, n^{aux}(1 - \bar{\mu}_k^{aux}))$ for all arms $k \in [K]$.
**for** $t = 1$ **to** $T$ **do**
    Sample $\theta_k^t \sim$ beta$(\alpha_k^t, \beta_k^t)$ for all arms $k \in [K]$.
    Play the arm $a_t = \arg\max_{k \in [K]} \theta_k^t$.
    Observe the payoff $r_{a_t}$.
    Update $(\alpha_k^{t+1}, \beta_k^{t+1}) \leftarrow (\alpha_k^t, \beta_k^t) + (r_{a_t}, 1 - r_{a_t})$ if $a_t = k$; update $(\alpha_k^{t+1}, \beta_k^{t+1}) \leftarrow (\alpha_k^t, \beta_k^t)$ if $a_t \neq k$.
**end for**
```

or 1200 in clicks on the tuning dataset. The Beta prior (α_k^0, β_k^0) implies a CTR mean of $\alpha_k^0/(\alpha_k^0 + \beta_k^0)$ and variance of $\alpha_k^0\beta_k^0/((\alpha_k^0 + \beta_k^0)^2(\alpha_k^0 + \beta_k^0 + 1))$. We use the mean and variance from the training data to initialize the priors to $\alpha_k^0 = 1.38$ and $\beta_k^0 = 96.11$ for all $k \in [K]$ in our experiment.

We now present the results on the performance of both LLM-TS and the standard TS algorithm on the test dataset in Figure A14. For comparison, we also show the performance of LLM-2UCBs and the other benchmarks. We also document the pairwise comparison of different algorithms and report the relative percentage reward improvement and the p-value from the t-tests in Table A3. We find that UCB-based algorithms perform better than TS-based algorithms in both the LLM-assisted version and the standard version. This observation is not uncommon because it is widely known that TS suffers from over-exploration (Min et al., 2020); in other words, UCB's upper confidence shrinks faster, leading it to explore more effectively). As a result, while it is possible to implement a Bayesian bandit such as an LLM-Assisted Thompson Sampling within our LOLA framework, at least in our setting, UCB-based algorithms perform better.

G.3 LOLA with Contextual Information

We now introduce two additional LOLA algorithms that leverage contextual (i.e., user-specific features) information.

For linear contextual bandits, we assume that at each round, we observe the context vector $x_{t,k} \in \mathbb{R}^d$ for each arm $k \in [K]$. This context information can include LLM embedding vectors and user-specific features, among other features. The random reward of arm k at round t is denoted by $r_{t,k}$. After pulling one arm a_t at each round, only the reward of that arm, r_{a_t}, is revealed. We make the linear realizability assumption that the expected reward is linear in the context vectors, which means $\mathbb{E}[r_{t,k}|x_{t,k}] = x_{t,k}^\top \theta^*$. The weight vector $\theta^* \in \mathbb{R}^d$ is unknown and needs to be learned. Additionally, we assume access to a historical dataset of size M, denoted by $\{(x_{t,k}, r_t)\}_{t=-M}^{-1}$. With these definitions, we now present our proposed Algorithm 4, named LLM-Assisted Linear Contextual Bandit. The LLM-LinUCB algorithm extends LinUCB Chu et al. (2011). Under the linear realizability assumption, this algorithm achieves asymptotically optimal regret, matching the lower bound up to logarithmic factors. Note that this algorithm does not have an auxiliary sample hyperparameter n^{aux} because the historical sample size M can automatically adjust the confidence bound via Gram matrix A, assuming the same linear reward function during the online learning phase.

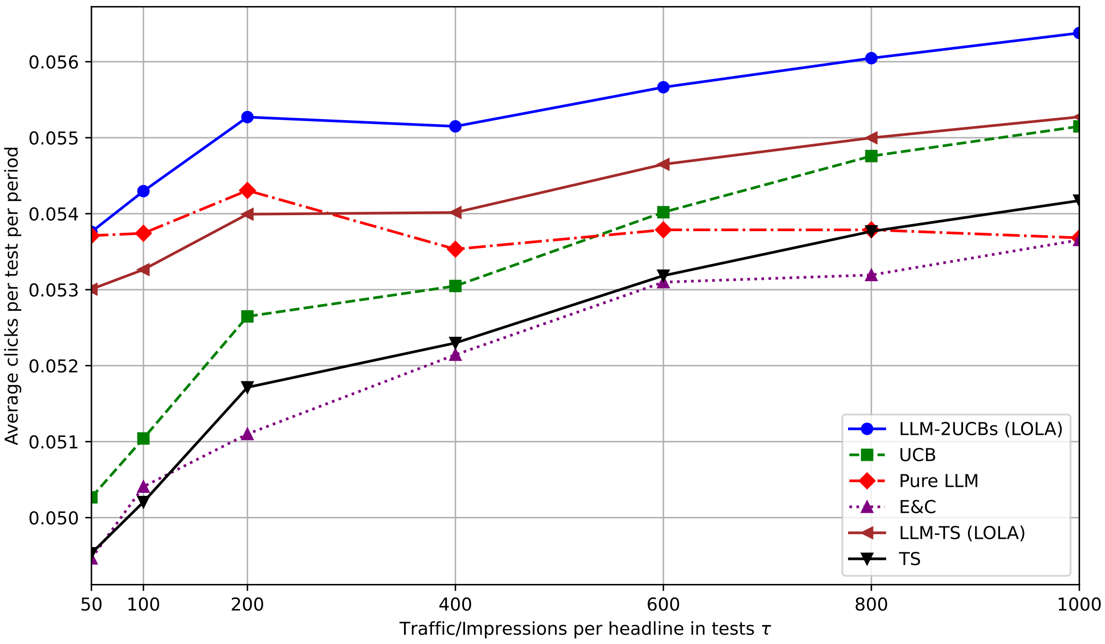


Figure A14: Average clicks per experiment per period under different time horizon multipliers. Note that the Y-axis captures the average clicks per test per period. For instance, if there is a test with two headlines conducted under $x = 100$, then the average click per period in this test is calculated as $(1 + 2)/100 = 0.03$. The Y value is simply the average of this number 0.03 over all tests. This measure scales well with the platform's total clicks in tests because headlines in different tests with different numbers of headlines take the same weight in this measure.

The algorithm proceeds as follows: During the classifier training phase, leveraging historical data, we prepare the matrix A and vector b to estimate the linear weight vector. We need an LLM-based embedding model to extract embedding vectors from the original text information. The hyperparameter fine-tuning is similar to LLM-2UCBs but with only the upper bound control parameter α. Specifically, one can use the bisection search algorithm to find the optimal α that maximizes the total reward on this fine-tuning dataset. The online learning phase remains the same as the standard LinUCB.

The advantages of LLM-LinUCB are its transparency due to the linearity assumption and ease of execution with only one hyperparameter, as well as the theoretical soundness of its optimal regret results. However, there are two issues with the linearity assumption. First, although we have shown that linear functions almost fully leverage text embeddings from OpenAI to achieve high accuracy (there is almost no difference compared to MLP), the embedding itself sacrifices a lot of information, as demonstrated by the significant outperformance of LoRA fine-tuning over embedding-based classification. Second, once concatenated with user contexts, it is unclear whether the linear function is a good approximation of the rewards model (and this requires further numerical study). In addition, LLM-LinUCB cannot accommodate LoRA fine-tuned open-source LLM, which is a complex nonlinear reward model.

To alleviate the limitations of the linear reward assumption and allow general LoRA fine-tuned open-source LLM with user-specific features, we introduce another general LLM-assisted contextual bandit algorithm (Algorithm 5), which extends the FAst Least-squares-regression-oracle CONtextual bandits (FALCON) algorithm proposed by Simchi-Levi and Xu (2022). This contextual bandit algorithm generalizes the linear realizability assumption to a more flexible framework, assuming that the true prediction model $f^* \in \mathcal{F}$. The functional class $\mathcal{F}$ can be very general and is not limited to linear models. This assumption is reasonable

Algorithm 4 LLM-Assisted Linear Contextual Bandit (LLM-LinUCB)

```
**Embedding-based Classifier Training Phase:** Calculate $A \leftarrow I_d + \sum_{t=-M}^{-1} x_{t,a_t} x_{t,a_t}^\top$, where $I_d$ is the $d$-by-$d$ identity matrix to avoid singular $A$, and $b \leftarrow \sum_{t=-M}^{-1} x_{t,a_t} r_t$.
**Hyperparameter Fine Tune:** $\alpha \in \mathbb{R}^+$ for controlling the upper bound.
**Online Learning Phase:** Initialize the number of periods $T$, number of arms $K$.
for $t = 1$ to $T$ do
    Get weight estimator: $\theta_t \leftarrow A^{-1} b$
    Observe $K$ context vectors, $x_{t,1}, x_{t,2}, \ldots, x_{t,K} \in \mathbb{R}^d$, including text embeddings, user-specific feature.
    for $k = 1, 2, \ldots, K$ do
        $\hat{r}_{t,k} \leftarrow \theta_t^\top x_{t,k} + \alpha \sqrt{x_{t,k}^\top A^{-1} x_{t,k}}$ (Computes the upper confidence bound)
    end for
    Choose action $a_t = \arg\max_{k \in [K]} \hat{r}_{t,k}$.
    Observe reward $r_{a_t}$.
    $A \leftarrow A + x_{t,a_t} x_{t,a_t}^\top$.
    $b \leftarrow b + x_{t,a_t} r_{a_t}$.
end for
```

and promising because LLMs have demonstrated their capability across a wide range of tasks. Essentially, $\mathcal{F}$ can be viewed as a class of LLM models, and the goal is to find the true LLM model f^* that performs best for our task, i.e., $f^*(x_{t,k}, k) = \mathbb{E}[r_{t,k} | x_{t,k}]$ for any $k \in [K]$ and $x_{t,k} \in \mathcal{X}$. Under this setting, the context information is not restricted to real numbers and can include more general data, such as text information. We now present the LLM-FALCON algorithm.

Algorithm 5 LLM-Assisted FAst Least-squares-regression-oracle CONtextual bandits (LLM-FALCON)

```
**LLM Training Phase:** Obtain the historical data $\{(x_{t,k}, r_t)\}_{t=-M}^{0}$, and construct an LLM $\mathcal{F}$ class for training.
**Hyperparameter Fine Tuning:** Confidence control parameter $\delta$.
**Online Learning Phase:** Initialize $T$, $K$, and the epoch schedule $0 = \tau_0 < \tau_1 < \tau_2 < \cdots$.
for epoch $m = 1, 2, \ldots$ do
    Set $\gamma_m = (1/30)\sqrt{(K\tau_{m-1})/\log(|\mathcal{F}|\tau_{m-1}/\delta)}$ (for epoch 1, $\gamma_1 = 1$).
    Compute $\hat{f}_m = \arg\min_{f \in \mathcal{F}} \sum_{t=-M}^{\tau_{m-1}} (f(x_{t,a_t}, a_t) - r_t)^2$ via the offline least squares oracle.
    for round $t = \tau_{m-1} + 1, \ldots, \tau_m$ do
        Observe contexts $x_{t,k} \in \mathcal{X}, \forall k \in [K]$.
        Compute $\hat{f}_m(x_{t,k}, k)$ for each arm $k \in [K]$. Let $\hat{a}_t = \arg\max_{k \in [K]} \hat{f}_m(x_{t,k}, k)$.
        Define
            $$p_t(k) = \begin{cases} \frac{1}{K + \gamma_m(\hat{f}_m(x_{t,\hat{a}_t}, \hat{a}_t) - \hat{f}_m(x_{t,k}, k))}, & \forall k \neq \hat{a}_t, \\ 1 - \sum_{k \neq \hat{a}_t} p_t(k), & k = \hat{a}_t. \end{cases}$$
        Sample $a_t \sim p_t(\cdot)$ and observe reward $r_{a_t}$.
    end for
end for
```

At the beginning of each epoch, we retrain the LLM using all available data up to that point, as detailed in Line 6. For simplicity, this algorithm assumes no optimization error during training, utilizing the offline least squares oracle instead. However, in practice, achieving zero optimization error is impossible, and one can incorporate this error into the final regret term using the regret triangle decomposition. Although this algorithm can handle original text information and fits into a general LLM model (not restricted to linear models), the primary challenge is the cost of training the LLM at each epoch because training a full LLM is typically time-consuming and costly. One potential solution, as investigated in our paper, is to fine-tune the LLM using LoRA instead of fully retraining it with new data. However, the empirical performance of this method remains unclear.

After getting an LLM trained on new data, then the key step in this algorithm is using the "inverse proportional to the gap" rule to balance exploration and exploitation. This rule is not new (Foster and Rakhlin, 2020; Simchi-Levi and Xu, 2022) and can effectively reduce a contextual bandit problem to a regression problem. At each round, it first identifies the empirically best arm $\hat{a}_t$ and then computes the gap in expected reward between this arm and the others. The sampling probability $p_t(k)$ is inversely proportional to this gap, meaning that a larger predicted reward $\hat{f}_m(x_{t,k}, k)$ results in a higher probability of being pulled/exploited. The gradually increasing parameter γ_m (as a sequential exploration and exploitation balancer) guarantees the algorithm to explore more in the beginning rounds and exploits more in later rounds.

Parameter τ	LLM-2UCBs vs. UCB	LLM-2UCBs vs. Pure LLM	LLM-2UCBs vs. E&C	LLM-2UCBs vs. LLM-TS	LLM-2UCBs vs. TS	UCB vs. Pure LLM	UCB vs. E&C	UCB vs. LLM-TS
50	6.95^{****}	0.09	8.69^{****}	1.42^{**}	8.54^{****}	-6.41^{****}	1.62	-5.17^{****}
100	6.38^{****}	1.03^{**}	7.72^{****}	1.94^{****}	8.15^{****}	-5.02^{****}	1.26	-4.17^{****}
200	4.98^{****}	1.78^{****}	8.16^{****}	2.37^{****}	6.88^{****}	-3.05^{****}	3.03^{**}	-2.49^{****}
400	3.96^{****}	3.02^{****}	5.76^{****}	2.10^{****}	5.45^{****}	-0.90	1.74^{*}	-1.79^{****}
600	3.04^{****}	3.49^{****}	4.83^{****}	1.86^{****}	4.66^{****}	0.43	1.73^{**}	-1.15^{***}
800	2.35^{****}	4.20^{****}	5.36^{****}	1.91^{****}	4.24^{****}	1.81^{***}	2.94^{****}	-0.44
1000	2.23^{****}	5.02^{****}	5.08^{****}	2.00^{****}	4.07^{****}	2.73^{****}	2.79^{****}	-0.23

Parameter τ	UCB vs. TS	Pure LLM vs. E&C	Pure LLM vs. LLM-TS	Pure LLM vs. TS	E&C vs. LLM-TS	E&C vs. TS	LLM-TS vs. TS
50	1.48^{*}	8.59^{****}	1.33^{**}	8.44^{****}	-6.68^{****}	-0.14	7.02^{****}
100	1.67^{***}	6.61^{****}	0.90^{**}	7.04^{****}	-5.36^{****}	0.40	6.09^{****}
200	1.80^{****}	6.27^{****}	0.58	5.01^{****}	-5.36^{****}	-1.19	4.41^{****}
400	1.43^{****}	2.66^{***}	-0.89^{**}	2.36^{****}	-3.47^{****}	-0.30	3.28^{****}
600	1.57^{****}	1.30	-1.58^{****}	1.13^{*}	-2.84^{****}	-0.16	2.76^{****}
800	1.84^{****}	1.12	-2.20^{****}	0.04	-3.28^{****}	-1.07^{*}	2.29^{****}
1000	1.80^{****}	0.05	-2.88^{****}	-0.90	-2.93^{****}	-0.96^{*}	2.03^{****}

Table A3: Percentage % reward improvement between two algorithms with significance levels: $^{*}\ p \leq 0.05$, $^{**}\ p \leq 0.01$, $^{***}\ p \leq 0.001$, $^{****}\ p \leq 0.0001$. For example, under the impression per headline equal to 50, the average click per test of LOLA and UCB is 0.053760 and 0.050267, respectively; so the relative improvement is calculated as $0.053760/0.050267 - 1 = 6.95\%$. Parameter τ represents the average impression per headline to scale the time horizon $T = \tau \times K$.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 208–214. JMLR Workshop and Conference Proceedings, 2011.

Qing Feng, Tianyi Ma, and Ruihao Zhu. Satisficing exploration in bandit optimization. *arXiv preprint arXiv:2406.06802*, 2024.

Dylan Foster and Alexander Rakhlin. Beyond ucb: Optimal and efficient contextual bandits with regression oracles. In *International Conference on Machine Learning*, pages 3199–3210. PMLR, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Blake Mason, Lalit Jain, Ardhendu Tripathy, and Robert Nowak. Finding all ϵ-good arms in stochastic bandits. *Advances in Neural Information Processing Systems*, 33:20707–20718, 2020.

Seungki Min, Ciamac C Moallemi, and Daniel J Russo. Policy gradient optimization of thompson sampling policies. *arXiv preprint arXiv:2006.16507*, 2020.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

David Simchi-Levi and Yunzong Xu. Bypassing the monster: A faster and simpler optimal algorithm for contextual bandits under realizability. *Mathematics of Operations Research*, 47(3):1904–1931, 2022.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, et al. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*, 2024.