Building reliable and transparent machine learning systems using structured intermediate representations

Giulio Zhou

April 2024

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David G. Andersen, Chair
Zachary Lipton
J. Zico Kolter
Byron Wallace (Northeastern University)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Machine Learning, Reliability, Transparency, Parametric Models

Abstract

Machine learning (ML) is increasingly used to drive complex applications such as web-scale search, content recommendation, autonomous vehicles, and language-based digital assistants. In recent years, these systems have become predominantly data-driven, often underpinned by deep learning models that learn complex functions end-to-end from large amounts of available data. But their purely data-driven nature also makes the learned solutions opaque, sample inefficient, and brittle.

To improve reliability, production solutions often take the form of *ML systems* that leverage the strengths of deep learning models while handling auxiliary functions such as planning, validation, decision logic, and policy compliance using other components of the system. However, because these methods are often applied post-hoc on fully trained, blackbox deep learning models, their ability to improve system reliability and transparency is limited.

In this thesis, we study how to build more reliable and transparent ML systems using ML models with *structured intermediate representations* (StructIRs). Compared to non-structured representations such as neural network activations, StructIRs are directly obtained by optimizing a well-defined objective and are structurally constrained (e.g., to normalized embeddings or compilable code) while remaining sufficiently expressive for downstream tasks. They can thus make the resulting ML system more reliable and transparent by increasing modularity and making modeling assumptions explicit.

We explore the role of StructIRs in three different ML systems. In our first work, we use simple probability distributions parameterized by neural networks to build an effective ML-driven datacenter storage policy. In our second work, we show that grounding text generation in a well-structured vector embedding space enables effective transformation of high-level text attributes such as tense and sentiment with simple, interpretable vector arithmetic. In our final work, we conduct human subject studies showing that the stationarity assumptions behind bandit-based recommender systems do not hold in practice, demonstrating the importance of validating the assumptions and structures underlying ML systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, machine learning (ML) has become ubiquitous. Sophisticated ML models (primarily deep neural networks) now drive the myriad digital applications and services that have become inescapable fixtures of modern life, including recommender systems, search engines, and language-based digital assistants. Compared to their traditional counterparts, modern ML models are largely data-driven and typically make little assumptions about the function being learned. This has enabled them to take advantage of recent advances in compute and data infrastructure to learn complex functions from patterns in massive quantities of data. Most recently, large language models (LLMs) such as GPT-3 [20], PaLM [28], OPT-175B [157], and BLOOM [125] containing hundreds of billions of parameters have been trained using exaflop-scale compute clusters. The growing capabilities of large deep learning models have shown that they can reliably learn general functions *end-to-end* given sufficiently large amounts of data and computation [66].

In the language of software systems, recent trends towards encapsulating the entirety of system functionality within a single model follows so-called "monolithic" system design. While monolithic systems may be appropriate for simpler (e.g., single-application) use cases, they are often insufficient for complex production-scale workflows that benefit from a more modular approach providing improved reliability, scalability, and transparency. Similarly, because ML systems are being deployed in increasingly complex and production-critical settings, there is a growing requirement for systems that consistently achieve a desired level of performance while doing so in a way that is auditable and accountable to its actions.

To address this challenge, researchers and practioners have been designing so-called *compound AI systems* [156]. In contrast to monolithic (single-model) systems, compound AI systems are often more expressive, reliable, and inspectable by being multi-component e.g., a question-answering pipeline that retrieves relevant documents, generates multiple candidate answers, and then validates them to ensure that answers are grounded in the retrieved source material. However, because they were not trained end-to-end, compound ML systems are not necessarily interoperable off-the-shelf. We therefore consider the following question: *Can we design reliable and transparent machine learning systems by giving compound AI systems the modularity and interoperability of traditional software systems?*

In this thesis, we present an initial step towards this vision. We propose training ML models to output *structured intermediate representations* (StructIRs) – including (but not limited to) vector embeddings, probability distributions, schematicized JSON, compilable code – that we show can be utilized as effective inputs into downstream systems. While there are many possible instantiations of compound AI systems, we consider (as the object of study in this thesis) a two-stage ML system that first applies a ML model ϕ to inputs x to produce a StructIR $z = \phi(x)$ that is then passed into an output process h to generate the desired outputs $y = h(z)$. We study two instances of this type of system: (1) a ML-driven datacenter storage policy that outputs storage decisions based on object lifetime estimates, and (2) a controllable language-generation system that outputs generated text conditioned on its exact vector representation in a high-dimensional embedding space. The main features of StructIRs are that they must be:

- The direct outputs of an optimization process on a well-defined objective.
- Structurally constrained with rules or criteria determining what constitutes a valid instance.
- (In terms of format,) Sufficiently general-purpose and expressive for many downstream use cases.
- Used as the primary or sole determinant of the contents of the output process.

Having demonstrated the promise of building ML systems with learned and validated StructIRs, the last part of the thesis focuses on a related question: *How do we ensure that the assumptions behind the StructIRs used in our ML system are valid?* To address this, we contribute an experimental framework for evaluating assumptions on human preferences in the context of bandit-based recommendation systems. Our thesis statement is as follows.

Thesis Statement: Machine learning systems can be designed to be more reliable and transparent by training the underlying ML models to output *structured intermediate representations* (StructIRs). We show that StructIRs can improve reliability by:

1. Introducing structural constraints into the representation that improve usability and compatibility with downstream systems.
2. Being more tractable to learn compared to end-to-end solutions.
3. Enabling more modular evaluation and design of machine learning systems.

In addition, StructIRs improve transparency by:

1. Ensuring that modeling assumptions about problems are made explicit.
2. Allowing issues with the generated outputs can be traced back to its corresponding structured intermediate representation.
3. Enabling compound AI system design through enhancing the mutually compatibility of its constituent components.

We also show that it is critical to ensure that the assumptions made about the StructIRs are within our systems are valid.

The remainder of this thesis is organized as follows:

- In Chapter 2, we provide general background on machine learning and deep learning models. We also discuss approaches for end-to-end problem solving and imposing structured representations.
- In Chapter 3, we show how to build effective and transparent ML policies for datacenter

storage systems by training natural language processing models to output StructIRs in the form of object lifetime probability distributions.

- In Chapter 4, we demonstrate that vector-to-text methods can be used to reliably transform high-level text attributes using simple, interpretable vector arithmetic within an existing learned high-dimensional embedding space.
- In Chapter 5, we outline an experimental framework that we use to empirically test the stationarity assumptions of bandit-based recommendation systems.
- In Chapter 6, we summarize our conclusions and discuss directions for future work.

We outline each of our case studies below.

1.1 Case Study 1: Effective Datacenter Storage Policies using Univariate Probability Distributions

Modern datacenter systems are highly complex and interconnected. When building web-scale applications, developers often opt to construct high-performance workflows from a host of cloud services - such as web endpoints, databases, data processing systems, and storage systems. These cloud systems achieve substantial improvements in performance and cost efficiency over their locally hosted counterparts through their enormous scale that enables holistic optimizations over workloads and infrastructure. While efficient and low-overhead, this design also introduces additional complexity in the form of performance variability. For developers, understanding the root cause of application slowdowns is challenging since they can come from any of the multiple (opaque) systems being used. In conjunction, these systems themselves also observe highly variable input behavior from the heterogeneous mix of datacenter workloads, complicating system management by rendering one-size-fits-all policies ineffective.

The above examples of performance variability highlight the importance greater visibility into as well as for datacenter systems. On one hand, developers have access to a tool known as *distributed tracing* for analyzing and debugging distributed workloads. Tracing collects metrics by interposing along the application request path through lightweight instrumentation. Assuming that the target systems have been appropriately instrumented, distributed tracing can cross reference application-level features (e.g. class of service) with fine-grained system information (e.g. the exact code path taken or database call.) On the other hand, system designers have limited options for making application- and workload-specific information accessible. While users can manually add hints (such as prefetching), it is a brittle process that leads to increased technical debt. Instead, in our first work, we demonstrate how ML can make effective use of high-level information already available in the system, in the form of distributed traces; incoming storage requests are tagged with key-value string pairs containing rich application-level information. We focus on the problems of caching and storage tiering within large warehouse-scale storage systems such as Colossus [45], and show how ML can be used to create effective storage policies.

Finally, we outline the challenges of applying ML in this setting and how incorporating parameterized structure allows our approach to do so successfully. We have discussed how datacenter storage and other systems are heavily interconnected, so even small performance regressions can have massive downstream effects. These systems are constantly making decisions to optimize performance criteria. While this suggests a feedback-driven approach such as reinforcement learning (RL), the resulting end-to-end learned solutions are often too sample inefficient [], brittle, and opaque for use in production systems. Rather, our approach uses ML to directly model the variability in the system using an approach based on survival anaylsis. Aggregating across read requests with the same observed features, we impose a Markov model on the time between repeated accesses to the same 128kB file block and fit parameters of a log-normal distribution using a deep learning model. We use a similar setup for storage tiering, where we instead model the lifetime and final size of entire files based on features in the file creation request. We use log-normal distributions since their parameters can be estimated in closed form and are more interpretable than that of similar distributions such as log-logistic and Weibull distributions. We find that these learned lifetime estimates can be utilized by simple hand-written policies to improve caching and storage tiering performance in simulation.

1.2 Case Study 2: Steerable Language Generation from Text Embedding Vectors

Over the past decade, systems for language generation have seen widespread development and adoption. As diversity of language generation applications has increased – from earlier use cases such as machine translation to current-day use cases such as code generation, automated writing assistants, dialogue systems, and language-based agents – so has the sophistication of the models that drive these systems. Compared to the structured, rule-based models that powered machine translation systems in the previous decade, deep learning models are statistical in nature and significantly more expressive, containing millions to billions of parameters that are optimized end-to-end in a data-driven fashion. These developments have led to substantial improvements in the quality and generality of modern language systems. Deep learning-based approaches to language generation, however, lack the performance guarantees of their predecessors and the ability to inspect the steps taken that led to the generated outputs. The proliferation of large, foundation-style models such as GPT-4, Claude, Gemini has furthered the trend towards machine learning systems becoming monoliths that are only testable via their end-to-end behavior.

Increasingly, practitioners and researchers are recognizing the limitations of such monolithic systems and are exploring multi-component solutions to language generation. For instance, a monolithic question-answering system must encapsulate knowledge and answer generation within a single model, which can lead to unreliable performance (e.g., if model knowledge is incorrectly learned or outdated) that is challenging for a system designer to diagnose. As a result, many recent systems have begun to rely on retrieval-augmented generation (RAG), i.e., allowing the language model responsible for text generation to access an external knowledge database represented as text embedding vectors. The resulting solution is often (1) more reliable by generating answers using external knowledge instead of relying on the model internals alone, and (2) transparent because there is an explicit connection between the model's outputs and the knowledge that it indexed. Compared to the (monolithic) end-to-end solution, this approach represents a class of alternative solutions that are more modular and easily tested.

In this section, we consider an even simpler example of a compound ML system that generates text x conditioned on a single embedding vector z, leveraging a recently developed approach by Morris et al. [103] known as `vec2text`. Given an existing, fixed embedding model ϕ, they train a conditional language model that learns to reconstruct the original text x from its embedding vectors representation $\phi(x)$. While Morris et al. [103] originally focused on assessing the privacy considerations behind vectors retain information about their originating text, while we instead ask the following question: *Can* `vec2text` *be used to create a reliable and transparent language generation system using StructIRs in the form dense vector representations of text?* We show that this is indeed possible, by leveraging `vec2text` to transform high-level semantic properties of text such as tense, sentiment, conciseness, and gender using simple vector arithmetic in the embedding space of the widely used OpenAI `text-embedding-ada-002` embedding model. We show that `vec2text` can reliably perform these transformations, outperforming baselines based on latent vector and activation steering, and does so transparently due to the direct connection between the generated output $\tilde{x}$ and its embedding representation $\phi(\tilde{x})$. We additionally present other findings about the embedding space, including the ability to effectively (and transparently) combine multiple transformations and also on the presence of gender-occupation associations. We make the case that the success of this approach is a consequence of the StructIRs used. Because embeddings (such as `text-embedding-ada-002`) are trained to encode high-level semantic relationships in a simple manner (linearly), the resulting ML system is able to generate high-quality text simply and without time spent on training or integration.

1.3 Case Study 3: Validating Stationarity in Bandit-based Recommendation Systems

In this chapter, we investigate the importance of validating the structural assumptions of the ML model that underlies a ML system. When designing a ML system, it is common to incorporate structural or simplifying assumptions into the model in order to improve its learning process and deployment efficacy. Assumptions may be based on known or idealized models of how the modeled system behaves, such as physics-based temporal modeling and multi-view geometry. Ultimately, structural and simplifying assumptions make learning more efficient and the learned model more reliable by adding inductive bias and invariances. However, imposing modeling assumptions that are too restrictive or unrealistic can lead to highly suboptimal and unpredictable performance.

We tackle this problem by empirically testing whether a common assumption made by bandit-based recommendation systems holds true in practice, i.e. that human preferences remain static. We believe that the benefits of this are two-fold. First, in some cases, it can substantially improve model learning that is otherwise degraded by model mis-specification. Second, incorrect modeling assumptions can also lead to unintended effects when deployed in production. For multi-armed bandit-based recommendation systems that assume static reward distributions, drifting preferences can complicate learning. For example, the UCB and epsilon greedy algorithms decay their learning over time, so changes from the original preferences may not be properly incorporated into the model, which also worsens user satisfaction. Additionally, the system's choice of recommendations (and the order in which it is presented) may also influence the user's preferences. Psychology studies have shown that for users of online music and video platforms, recommendations can influence their internal states such as inducing anchoring in their reported preferences, as well as altering their moods and opinions.

Our setup is as follows. We conducting randomized controlled trials on human subjects from Mechanical Turk using a simulated content recommendation system. We set up a bandit-based recommendation system based on the task of reading comics where each arm represents the category of comic being read. To test for the presence of non-stationarity in the user preferences, we randomly assign a subset of users to one of two fixed sequences of comics: CYCLE and REPEAT. We then perform multiple hypothesis testing over the differences in mean reward for each category using a permutation test. We find that there is a statistically significant difference between the two sequences among all of the categories.

Chapter 2

Background

2.1 Deep Learning Models and Representation Learning

2.1.1 Heuristics to Machine Learning and Deep Learning

2.2 Machine Learning Systems

2.2.1 Compound ML Systems

To address the shortcomings of these large monolithic ML systems, there has been a shift towards the development of so-called *compound ML systems*. Compared to monolithic systems that encapsulate all functionality within a single model, compound ML systems instead compose multiple ML- and software-based components into an end-to-end system. As an illustrative example, search and question-answering are most commonly handled by compound ML systems in the form of *Retrieval-augmented generation* (RAG). While single-model systems dedicate model capacity to both knowledge and text generation, RAG systems instead augment text generation models with an external knowledge base. The resulting design is more modular and adaptable to novel content without constant re-training. Notably, these developments closely parallel the evolution of monolithic, single-machine software architecture to multi-component, modular datacenter systems that has improved their scalability, reliability and transparency.

Other examples of compound ML systems include AlphaCode] and AlphaGeometry], which each use LLMs to generate numerous candidate solutions (to coding and geometry problems respectively) that are then verified by downstream systems.

Chapter 3

Effective Datacenter Storage Policies using Univariate Probability Distributions

3.1 Introduction

Modern data centers contain a myriad of different storage systems and services, from distributed file systems [45, 57, 133, 153], to in-memory caching services [39] and databases [30, 138]. These services typically operate behind an RPC abstraction and are accessed by workloads that are composed of interconnected services communicating through RPCs [43].

Storage services continuously make *decisions* that aim to optimize metrics such as cache hit rate or disk footprint. To make these decisions, the systems need to make *predictions* about future workload and system behavior. For example, caches admit objects based on their likelihood of future access [14, 61], and block allocators reduce fragmentation by colocating allocations of comparable lifetime [72].

Traditionally, storage systems rely on heuristics for these decisions, such as LRU replacement policies or best-fit allocation. These heuristics exploit statistical workload properties like temporal or spatial locality, but are unable to leverage application-level signals, such as whether or not a request belongs to a temporary file. While systems can communicate hints to the storage system (e.g., using prefetch commands or non-temporal stores), manually assigning these hints is brittle, work-intensive and incurs technical debt. As such, they are most commonly used in highly tuned workloads. To apply such optimizations to the long tail of data center workloads [65], we need to automate them.

We observe that in many cases, high-level information is already available in the system, as part of distributed tracing frameworks [9, 134] and resource managers [31, 110] that are widely deployed in data centers. Distributed traces, job names, permission names, etc. are generated automatically as part of the system's regular operation and encapsulate human-defined structure and information.

In this work, we are looking at a specific instance of this approach using a deployed production distributed tracing framework, Census [108]. Census provides a tagging API that applications use to generate arbitrary key-value pair strings (Census Tags) that are automatically propagated with outgoing requests. These tags can be used to understand complex workload

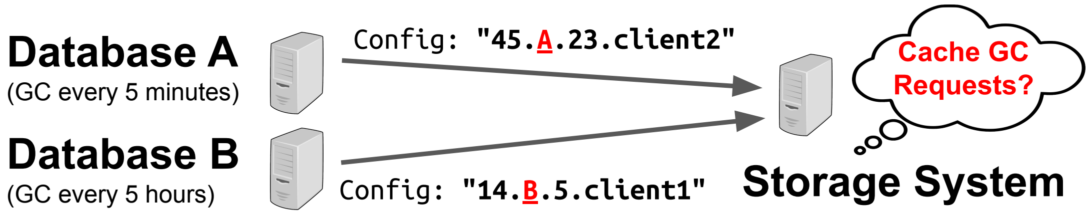


Figure 3.1: Distributed tracing tags contain unstructured application information that can be leveraged to make storage predictions.

interactions, and for resource accounting. A side effect is that incoming requests now come with rich context that encodes the path taken to the storage system, which the system can leverage.

However, this data does not always have an explicit schema or directly encode the information required by the storage system. For instance, consider a hypothetical example of databases with two different configurations A and B, which are listed in a configuration string attached to each storage request (Figure 3.1). A has a garbage collection interval of 5 minutes while B has 5 hours. A caching service could leverage this information by only caching requests from the service with configuration A. However, this information is not readily available: The service needs to know that it needs to check the configuration string for the presence of A or B in a particular location, and which requests to drop.

Instead of explicitly encoding these rules, we learn them from historical trace data. We present several techniques, ranging from lookup tables to neural networks that leverage recent progress in natural language processing. A key challenge is that models become stale over time and do not transfer to new settings (e.g., a new storage system or cluster). The reason is that the model jointly has to learn 1) how to extract information from distributed traces and 2) how this information translates to predictions in a storage system. If the storage system changes, both need to be relearned from scratch. We therefore introduce a model that can be used in a multi-task learning setting, where a model can be used as a building block in different task-specific models.

We make the following contributions: 1) We demonstrate the connection between distributed tracing and storage-layer prediction tasks (Section 3.2) and show that strong predictive performance relies on leveraging the latent structure of unstructured distributed traces (Section 3.3). 2) We show that several important (and traditionally separate) storage tasks - such as cache admission/eviction, file lifetime, and file size prediction - can be learned by the same models from application-level features. 3) We present models of increasing complexity (Section 3.4) that represent different deployment strategies, and analyze their trade-offs. 4) We show that our models are robust to workload distribution shifts and improve prediction accuracy by over non-ML baselines, for a range of storage tasks, improving both a caching and an SSD/HDD tiering task substantially in simulations based on production traces (Section 4.5).

3.2 Background & Related Work

Data Center Storage Systems. We use a broad definition of what constitutes a storage system. We consider any service within a warehouse-scale computer that holds data, either on persistent storage (e.g., databases, distributed file systems) or in-memory (e.g., key-value stores). Such services exist at different levels of the storage stack: managing physical storage (e.g., storage daemons in Ceph [153] or D file servers [127]), running at the file system level (e.g., HDFS [133]) or storing structured data (e.g., Bigtable [23]). One storage system may call into another. We assume that requests to the service are received as RPCs with attached metadata, such as the request originator, user or job names, priorities, or other information.

Prediction in Storage Systems. Storage systems employ various forms of prediction based on locally observable statistics. These predictions are often implicitly encoded in the heuristics that these systems use to make decisions. For example, LRU caches make admission decisions by implicitly predicting diminishing access probability as the time since previous access increases, while FIFO-TTL caches evict objects assuming a uniform TTL.

While such policies can model a broad range of workload patterns, they have limitations. First, a single heuristic may have difficulties modeling a mixture of workloads with different access properties, which it is more likely to encounter in warehouse-scale computers where storage services receive a diverse mix of requests resulting from complex interactions between systems. Second, they are limited by their inability to distinguish between requests based on application-level information. For example, while traditional caching approaches can distinguish between read and write requests, they do not typically distinguish based on what application-level operation the request corresponds to.

Application-Level Information in Systems. Using high-level features in storage systems is not a new idea. However, most mechanisms require that the application developer explicitly provides hints. A less explored alternative is to extract such high-level information from the application itself. Recent work has demonstrated a similar approach for cluster scheduling [110], by predicting a job's runtime from features such as user, program name, etc. While cluster schedulers have a host of such features available at the time of scheduling a job, exploiting such information in storage systems is more challenging, since the predictive features are not readily available. For example, a storage request may be the result of a user contacting a front-end server, which calls into a database service, which runs a query and in turn calls into a disk server. Features may have been accumulated anywhere along this path.

The same challenges that make it difficult to reason about storage requests make it difficult to monitor, measure and debug distributed systems in general. For this reason, data centers have long employed distributed tracing frameworks [9, 134], which track the context of requests between systems. Distributed traces thus present an opportunity to leverage application-level features for predicting workload behavior. Unfortunately, the data gathered by these systems can be difficult to exploit, due to its high dimensionality and unstructured nature.

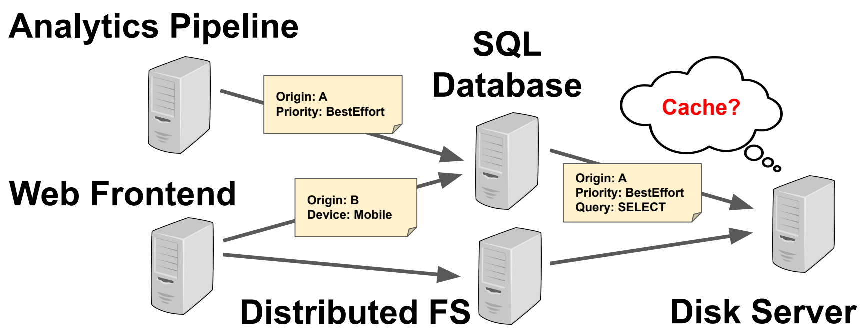


Figure 3.2: Census tags are free-form key-value pairs that are added by services and propagated with subsequent requests.

ML for Data Center Systems. The promise of ML for storage-related tasks is its ability to learn useful representations from large amounts of unstructured data. For example, Gan et al. [44] showed that it is possible to use traces to predict QoS violations before they occur. Different techniques have been proposed. For example, cheap clustering techniques [31, 110] and collaborative filtering [34] have been shown to work well for cluster scheduling while the aforementioned work on distributed traces relies on LSTM neural networks [55]. It is important to distinguish between ML for predictions/forecasting and ML for decision making. Prior work on applying ML to storage systems has sought to optimize the latter, such as learning caching policies [73, 91, 137]; these works improve upon heuristics while using conventional features. In contrast, we use ML to enable the use of more complex, application-level features.

Connection to Multi-Task Learning. Storage systems in data centers feature a wide range of settings. For example, caches within different systems behave differently, which means that their predictions differ as well. Decisions can also differ across database instances, or based on the hardware they run on. As a result, prediction in storage systems does not require training a single model but a myriad of them. Some systems even need to make multiple decisions simultaneously (e.g., lifetime and file size on file creation). This indicates that it is beneficial to share models between tasks, an approach known as multi-task learning (MTL).

There has been a large amount of work on MTL. Many advances in areas such as natural language processing and computer vision have come from using large amounts of data to learn general models that transfer better to new tasks. Among NLP's recent successes are the learning of general Transformer models [147], such as GPT-2 [114] and BERT [35].

Usually, MTL datasets do not have a complete set of labels for each task, but are often multiple datasets (with possibly disjoint task labels) that share a common input feature space. As such, MTL is a natural fit for learning multiple storage tasks from shared distributed traces.

3.3 Workload Analysis with Census

In this section, we describe how the information contained in distributed traces relates to prediction tasks in storage systems, and analyze their statistical properties.

Key	Example Values	Cardinality	Description
[redacted]Call	COMPACT(MINOR) — COMPACT(MAJOR) — BACKUP	Low	Request is from a particular DB operation
[redacted]CacheHit	hit — miss	Low	Whether a request was cached
[redacted]_action	EditQuery — GetIds — UpdateConfig	Medium	A particular operation from a service
user_id *	[redacted]-pipeline-services-low — [redacted]-jobs — local_[redacted]	High	A particular user group (free form)
JobId *	datacenterABC.client-job-5124.v521aef *	High	A particular job name (free form)
[redacted]Table	[redacted]-storage.[redacted]-local-[redacted].[redacted].[redacted].4523.index	High	Name of a table a query applies to
[redacted]TxnTag	AsyncService-Schedule-[redacted] — DELETE-LABEL — EDIT-[redacted]	High	Free form description of an operation

Table 3.1: Examples of Census tag features found in production distributed traces (adapted* and/or obfuscated).

3.3.1 Distributed Traces and OpenCensus

Data center applications are composed of services that communicate via message passing [43]. Services often have multiple clients using them (e.g., different services accessing a database) and rely on multiple different downstream services (e.g., the database service might connect to storage servers and an authentication service). This makes analysis and resource accounting challenging: Should a request to a database be attributed to the database or one of the upstream services using it?

Census [108] is a distributed tracing library that provides insights into such systems. It tags requests as they travel through the system (Figure 3.2). Census allows services to set key-value pairs (*Census Tags*) that are automatically propagated and allow a service to determine the context of each request that it receives (e.g., for resource accounting). These tags are set through an API by the service developers themselves, while being oblivious to tags added by downstream services. One of the insights of our work is that this same information represents powerful features for reasoning about the distributed system: Existing tags already capture properties of the workload that a programmer deemed important, and programmers could select new tags based on what they believe would be predictive features.

Some examples of Census tags are shown in Table 3.1 (obfuscated but based on real values). While this captures some common cases, this list is not exhaustive. Some Census Tags have low cardinality (they either take on a small number of values or their main information is in their presence), while others (such as transaction IDs, jobs or table names) have very large cardinality (sometimes in the tens of thousands). A human could sometimes manually write a regular expression to extract information (e.g., "[redacted]Table" might be split by "." and "-" characters and the first entry refers to a user group), but as Census tags are maintained by services themselves, there is no guarantee that they are not going to change. For storage services to make assumptions on any particular structure of these tags is inherently brittle.

We also noticed that the same tag does not always follow the same schema. For example, the "[redacted]TxnTag" shows a mix of different schemas depending on which service set the tag. Other tags feature a mix of capitalized/non-capitalized values, different characters to delineate different parts of the string, etc. This high degree of variance makes it difficult to manually extract information from these tags consistently.

3.3.2 Prediction Tasks

We now present prediction problems in storage systems that can benefit from high-level information.

File access interarrival time for caching. Predicting the time of the next access to an entry allows a cache to decide whether to admit it [14, 61], and which block to evict (e.g., a block with a later time). We focus on caching fixed 128KB blocks in a production distributed file system, which is either accessed directly or used as a backing store for other storage systems, such as databases. We ignore repeated accesses under 5 seconds, to account for local buffer caches.

File lifetime until deletion. File lifetime predictions are used in different contexts [72]. They are used to select storage tiers (e.g., for transient files) and can be used to reduce fragmentation. For example, some storage technologies have large erase units (e.g., SMR, NAND flash) and some storage systems are append-only [23, 109]. Placing data with similar lifetimes into the same blocks minimizes wasted bytes, write amplification, and compaction.

Final file size. Knowing the final size of a file at the time it is allocated can improve allocation decisions. For example, it can help disk allocators pick the best block size.

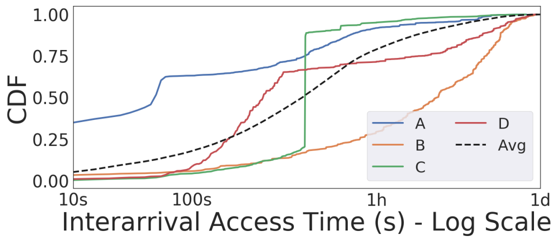
Read/write ratio. Predicting the ratio of read vs. write operations is helpful for placing data. Read-only files may be candidates for additional replication while write-only files may be best stored in a log structure. This prediction can also help pick a storage medium (Section 3.6.4).

This list is not exhaustive. Other tasks that are not explored in this work include 1) Resource demand forecasting when deploying a new mix of workloads (e.g., when bringing up a new cluster of machines), by recording a small number of samples characterizing the mix of workloads and then using a model to extrapolate the overall usage, and 2) Predicting workload interference (e.g., because both are I/O heavy).

3.3.3 Analyzing Census Tag Predictiveness

After introducing the prediction tasks, we now demonstrate that Census tags are predictive of some of these tasks.

Dataset. We analyze Colossus [127] file system traces sampled from 5 different clusters at Google. The data is longitudinally sampled at a per-file granularity. Our traces contain over a trillion samples per cluster and we are analyzing traces from a period of three years. The clusters contain different proportions of various workload types. All our requests are sampled at the disk servers backing the distributed file system and contain file metadata as well as Census Tags associated with each request. Note that these disk servers back other storage services, such as databases.



((a)) Overall and per-tag value interarrival time CDFs.

Task	Entropy	Cond.
Interarrival Time	4.748	3.474
File Lifetime	7.303	6.575
Final File Size	3.228	2.538
R/W Fraction	1.874	1.476

((b)) Overall and per-tag value interarrival time CDFs.

Figure 3.3: Conditioning the distribution on Census tags significantly reduces entropy, indicating that they are predictive.

Features. Broadly, the features provided through Census Tags fall into four categories: 1) Census tags that indicate a particular category of request (e.g., a DB operation), 2) numerical information (e.g., an offset), 3) medium and high cardinality labels that can contain unstructured data (e.g., project IDs, table names, etc.) and 4) high cardinality labels that may or may not be predictive (e.g., timestamps or transaction numbers). We are interested in the predictiveness of these features. Note that there is information about requests that these features do not capture. For example, we only consider one request at a time.

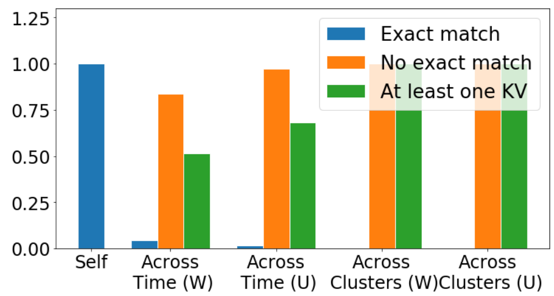
We can phrase our prediction problems as follows: Given a set of Census Tags and their associated values $X = \{x_1, x_2, \ldots, x_n\}$ where x_i is the ith (unordered) key-value string pair, predict a label Y (e.g., interarrival time, lifetime, etc.). We refer to X as a *Census Tag Collection (CTC)*.

Entropy Analysis. To measure the predictive ability of Census Tags, we look at the distribution of values for each of the tasks we aim to predict (e.g., interarrival times) and compare this distribution to the same distribution conditioned on different values of particular Census Tags. Figure 3.3(a) shows an example where we conditioned the distribution of interarrival times on a particular Census tag “[redacted]Key”, which describes what type of operation a request belongs to. We show distributions for four arbitrary values of this tag.

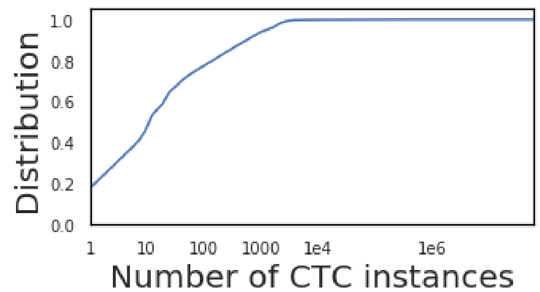
There is a visible difference between the distributions depending on the specific value, and the average distribution (the dotted line) captures neither of them. A way to measure this effect more formally is by computing the information entropy of the overall distribution (shown in Figure 3.3(b)) and compare it to the conditional entropy (the weighted average over the entropies when conditioning on Census tag values). The difference between the two is known as the mutual information (or information gain), which measures the predictiveness of Census Tag collections for the distribution.

Transferability of Census Tags. In order to use Census tags in predictions, we need to show that the information they provide transfers – i.e., values recorded in one setting can be used to predict values in a different setting. We are interested in two particular types of transferability:

1. **Across time:** We want to be able to use past traces to make predictions months or years in the future.



((a)) How many CTCs match across time and clusters.



((b)) How often each CTC appears in the data set (one cluster).

Figure 3.4: Transferability (a) and cardinality (b) of Census tags.

2. **Across clusters:** We want to use traces recorded in one cluster to make predictions in other clusters.

For predictions to be transferable, traces must either share features or have a similar latent structure (e.g., there exists a similar relationship between the keys and values even if they are named differently). To analyze transferability, we conducted a study comparing the keys and values found in two different clusters and between traces 9 months apart (Figure 3.4(a)). We find that 1) only a small fraction of requests have a CTC that occurs exactly in the original trace, but 2) most requests have at least one key-value pair that was seen in the original trace. This is true both across time and across clusters, and indicates that an approach that only records CTCs in a lookup table will degrade over time and is of limited use across clusters. Meanwhile, it shows that complex approaches can potentially extract more information.

High-Cardinality Tags. One example of tags that do not transfer directly are high-cardinality keys capturing information that changes over time or between clusters. For example, new accounts or database tables are added over time and different clusters host different workloads. Tags that directly include these identifiers as values will therefore differ. This is visualized in Figure 3.4(b) which plots the number of times each CTC is observed in a 1.4B entry trace. of CTCs are observed only once and , pointing to high-cardinality keys.

However, many of these tags can still contain information. For example, a username may be composed of a particular system identifier and a prefix (e.g., "sys_test54" vs. "sys_production") and table names often have hierarchical identifiers (e.g., a format such as "type.subtype.timestamp"). Only using exactly matching strings would therefore lose important information. We need to extract information from within these strings, which resembles natural language processing tasks. Such techniques enable proper information sharing between known values as well as generalization to new values that have not been seen before. Of course, there are also high-cardinality keys that carry little information – e.g., unseen UIDs. This has similarities to ML applied to code [68, 132], where tokens are often highly specific to the context in which they appear.

3.3.4 Distribution-based Storage Predictions

Intuitively, we would expect a predictor for the storage prediction tasks from Section 3.3.2 to predict one value for Y (e.g., the expected lifetime of a file) given a CTC X. However, there is inherent uncertainty in these predictions: 1) features do not always capture all details in the system that determine the file's lifetime, and 2) effects outside the control of the system, such as user inputs, affect the predictions. For many predictions, there is not a single value that we could predict that is correct most of the time. Similar to the work by Park et al. [110] on cluster scheduling, we therefore predict a *probability distribution* of values. This distribution can then be consumed by the storage system directly, similar to EVA [14]: For example, a cache could evict a cache entry with a high variance in its distribution of interarrival times in favor of an entry with low interarrival time at low variance.

To perform distribution-based predictions, data within the traces needs to be pre-aggregated. Specifically, we need to take all entries in the trace with the same X and compute the distributions for each of the labels Y that we want to predict (interarrival times, lifetimes, etc.). To do so, we can collect a histogram of these labels for each CTC. We note a large skew: Some CTCs appear many orders of magnitude more often than others, and of entries show up only once (Figure 3.4(b)). This can be explained by two effects: 1) Some systems account for a much larger fraction of requests than others, and 2) The lower the cardinality of Census tags set by a system, the lower the number of different CTCs associated with this system.

Fitting Lognormal Distributions. One approach to use the histograms for input features is to use them in a lookup table, which covers low-cardinality cases. However, as we have seen, some Census Tags have high cardinality and we therefore need to predict them using models that have the ability to generalize to previously unseen values. For these tags, we therefore need to represent the output distribution in a way that we can train a model against.

Gaussians (or mixtures of Gaussians) are often used to model this type of output distribution. For example, they are used for lifetime prediction in survival analysis [37]. In particular, we consider lognormal distributions and show that they are a suitable fit for our data. They are a popular choice for modeling reliability durations [104]. In contrast to other similar distributions (such as Weibull and log-logistic), the parameters that maximize the lognormal likelihood can be estimated in closed form. Figure 3.5 shows examples of fitting lognormals to the pre-aggregated distributions for several CTCs. To measure how well the fitted distributions match the real data, we use the Kolmogorv-Smirnov (KS) distance, which measures the maximum deviation between CDFs. The overall KS distance of fitting lognormals to our data is .

3.3.5 Case Study: Database Caching Predictor

We demonstrate the insights from this section using a database system as an example. One Census tag associated with this system indicates the high-level operation associated with it (Figure 3.3(a)). This information can be used to make decisions about admission and eviction. For example, consider values A, C and D of this particular Census tag. While the average (dotted) CDF of interarrival times for requests increases slowly (note the log scale), indicating that the interarrival time is difficult to predict, requests with values A/C/D are more predictable: The vertical jump

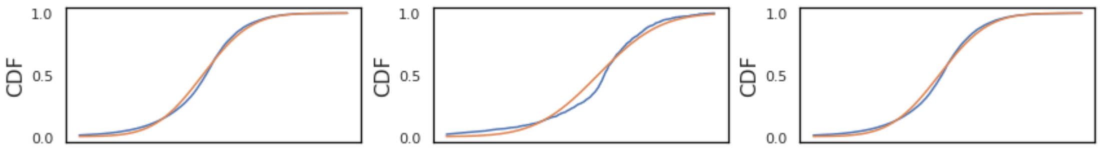


Figure 3.5: Fitting lognormal distributions to CTC CDFs.

in C shows that 3/4 of requests with this tag have an interarrival time of 15 minutes, indicating it has a periodicity of 15 minutes. Meanwhile, we see that 2/3 of requests for A take less than 1 minute before they are accessed again, and for D the same is true for a 5 minutes interval.

We can exploit this information in a caching policy that does not evict these requests for the first 1, 5 and 15 minutes after their last access. Afterwards, we treat them the same as other requests. We can also do something similar for values such as B where the distribution shows that interarrival times are much longer than for other requests. For example, we could avoid admitting these entries to the cache at all, or prioritize them for eviction.

3.4 Machine Learning for Census Tags

We now demonstrate a set of learning techniques to achieve transferability across clusters and over time. We assume that all models are compiled and directly linked into the storage server, running either on the CPU, or on an accelerator such as a GPU or TPU. When a request is received by a storage system, its CTC is represented as an unordered set of string key-value pairs. The prediction problem is formally defined as follows: Given a CTC X, predict the parameters of its lognormal distribution for a given task, $Y = (\mu_Y, \sigma_Y)$.

3.4.1 Lookup Table Model

The simplest prediction approach is a lookup table (Figure 3.6(a)) where a canonical encoding of the CTC is used to index a static table that maps CTCs to Y. The table is “trained” by collecting the target distribution histograms from a training set, pre-aggregating them, and computing the mean and standard deviation of a lognormal distribution that fits the data. CTCs are encoded by assigning each unique key and value in the training set an ID and looking them up at inference time. Keys in the CTC are sorted alphanumerically, ensuring that the same CTC always results in the same encoding.

CTCs not found in the table can be handled by substituting the overall distribution of the training set. As shown in Section 3.3.3, the entropy of this set is much larger than the entropy conditioned on a particular CTC (and is therefore not very predictive), but represents the best we can do. Note that the lookup table can become very large, and it is often necessary to remove rare CTC entries. There are different ways to implement such a lookup table. For example, it could be implemented as a hashtable or represented by a decision tree [123], which is an equivalent but potentially more compact representation.

3.4.2 K-Nearest Neighbor Model

Improving upon how the lookup table handles *unseen* CTCs, the k-nearest neighbor approach (Figure 3.6(b)) makes predictions for these entries by combining predictions from CTCs that are close/similar. We implement an approximate k-NN method that uses as its distance metric the number of differing Census Tags between two CTCs. We encode CTCs as a sparse binary vector where each entry denotes whether a particular Census Tag key-value pair is present. Distance can thus be cheaply computed as the squared L2 distance between sparse binary vectors (which can reach a dimensionality of millions). This approach allows us to make use of existing approximate nearest neighbor libraries that are highly optimized for sparse vectors. We choose K=50 in our experiments, since binary vectors may have many neighbors of equal distance. For instance, a CTC that has a different value for one Census Tag may get matched against a number of CTCs that have distance 2. To compute the predictions, we aggregate the chosen nearest neighbors. The mean μ_Y is simply the weighted average over the means of the individual neighbors. The standard deviation σ_Y is computed by summing two components: (1) the weighted average over the variance of each neighbor, and (2) the weighted squared distance between the individual means and the overall mean.

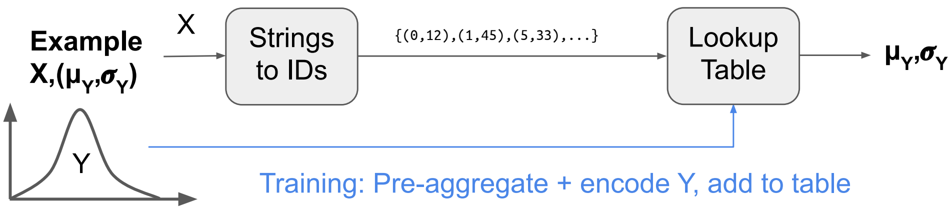
This approach resembles strategies that have been used in cluster scheduling [31, 110]. In contrast to a lookup table, it has more generalization ability, but it is still unable to extract information from high-cardinality Census tags. Imagine a tag where values are of format <query-type>.<timestamp>. Here, "query type" captures information that we want to extract. Since "timestamp" will take on a different value for every request, each entry will result in a different CTC. This, in turn, means that 1) the lookup table grows very large and 2) each entry only has a single data point associated with it. Instead of a histogram of values, the "distribution" associated with this CTC is therefore a single point mass with $\sigma = 0$. This makes it impossible for the model to generalize, since the nearest neighbor approach has no way of knowing that the different values are identical except for the timestamp.

3.4.3 Neural Network Model

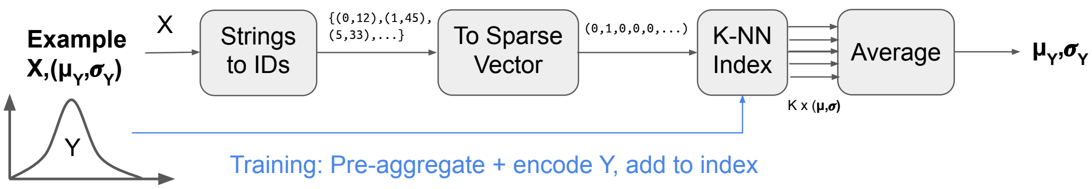
Handling these high-cardinality cases necessitates a model that can parse the strings that constitute the key-value pair. While a nearest neighbor approach can learn simple connections between predictions and Census tags (e.g., "if tag A has value B, the file is short-lived"), it cannot learn more complex and non-linear interactions (e.g., "if tag A has an even number as value, then the file size is small"). To push the limits of learning these more complex connections, we use a neural network-based approach. Note that in practice, this neural network would not run at every prediction but be used as a fall-back for lookup table entries where no example can be found (and therefore runs rarely).

A simple approach would be to encode keys and values as IDs (similar to the lookup table), feed them into a feed-forward network, and train against the Y from pre-aggregation. However, this approach still has no capability to generalize to unseen values nor high-cardinality keys that only have a single data point associated with them. We address these problems by combining two approaches:

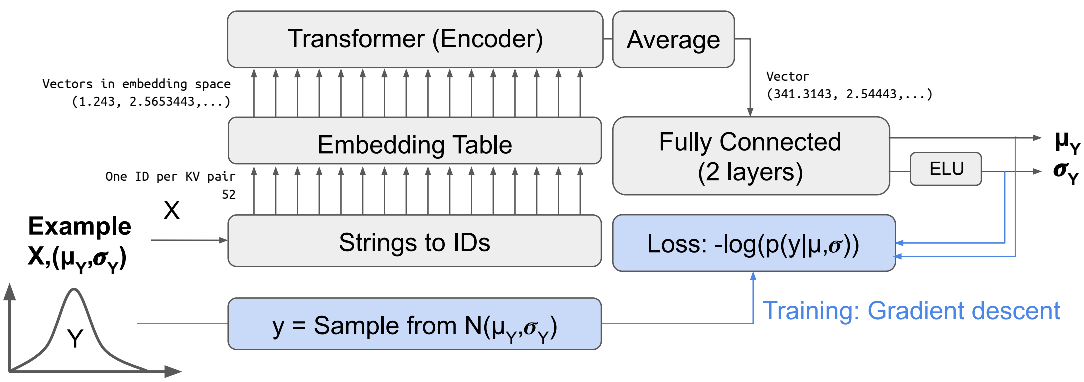
1. We build on recent advances in natural language processing to train networks operating on



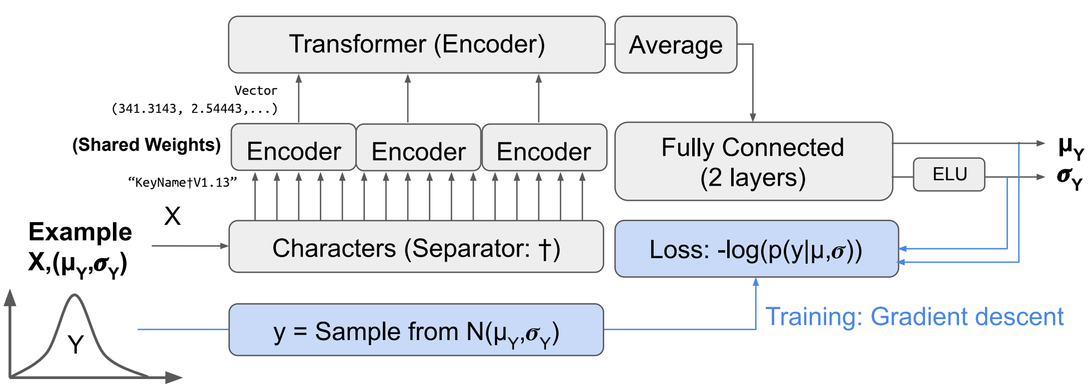
((a)) Lookup Table Model



((b)) Nearest Neighbor Model



((c)) Embedding-based Transformer



((d)) Hierarchical Raw Strings Transformer

Figure 3.6: The model architectures that we explore in our work (blue signifies training).

raw strings. Specifically, we use a Transformer [147] model that uses an attention mechanism to consume a sequence of inputs (e.g., character strings comprising each Census tag) and maps them to an embedding (i.e., a learned encoding).

2. To handle CTCs with a single point, we do not train against (Y_μ, Y_σ) directly, but use Mixture Density Networks [16] to let the model fit a Gaussian.

The neural network architecture is based on typical models used in NLP to process character and word tokens. We present two versions: 1) an embedding-based version that resembles the approach above of feeding token-encoded key-value pairs directly into the model, and 2) an approach that parses raw strings of key-value pairs. The model architecture is similar in both cases and relies on learning *embedding representations* [100], learned mappings from a high-dimensional input to a latent representation in some other – usually lower-dimensional – space.

Embedding-Based Transformer Model. For this version (Figure 3.6(c)), we start from a CTC $X = \{x_1, x_2, \ldots, x_n\}$ where x_i is the ith (unordered) key-value string pair – encoded as one-hot encoded vectors based on their IDs – and pass each x_i through a single embedding layer $\phi : N \rightarrow R^m$ to create a set of embedding vectors $V = \{\phi(x_1), \phi(x_2), \ldots, \phi(x_n)\}$. V is then passed into the Transformer encoder $M : R^{n\times m} \rightarrow R^{n\times m}$ and its output is averaged to produce the shared output embedding $S = \sum_{i=1}^{n} M(Y)_i$ where $S \in R^m$. Finally, this output embedding is passed through an additional 2-layer fully connected network to yield the outputs $Y = (\mu_Y, \sigma_Y)$. The last layer producing σ uses an ELU activation (specified by Mixture Density Networks).

Hierarchical Raw Strings. This version (Figure 3.6(d)) operates directly on raw strings, where each character is encoded as a one-hot vector of dimensionality 128 (127 characters and one special character to separate key and value). Each key-value pair x_i is encoded as a sequence of such one-hot vectors ($x_i \in R^{k_i\times 128}$), and the characters are passed through an embedding layer, yielding an $\phi(x_i) \in R^{k_i\times m}$, where k_i is the length of the i-th key-value pair. Each $\phi(x_i)$ is then passed through a Transformer encoder – all these encoders' weights are shared (i.e., this encoder learns how to parse an individual key-value pair). The outputs of these encoders are then passed into another encoder, which now aggregates across the different key-value pairs (i.e., it learns connections between them). As before, the output is then averaged and passed through two fully-connected layers.

Mixture Density Networks. Because the goal is to predict the distribution associated with each CTC, we must choose a loss function that allows the model to appropriately learn the optimal parameters. Consider if we used squared distance to learn the mean and standard deviation of a log-normal distribution. While squared error may be appropriate for learning the mean, it is not for the standard deviation. For instance, squared error is symmetric, and underestimating the standard deviation by 0.1 has a much larger effect on error than overestimating by 0.1. Additionally, a model trained with squared error will not learn the correct standard deviation from an overpartitioned dataset (e.g., if all CTCs had $\sigma = 0$, the model would learn $\sigma = 0$).

Mixture Density Networks [16] were designed to address this problem. Instead of fitting (μ_Y, σ_Y) directly to the label, the predicted (μ, σ) are used to compute the likelihood that the

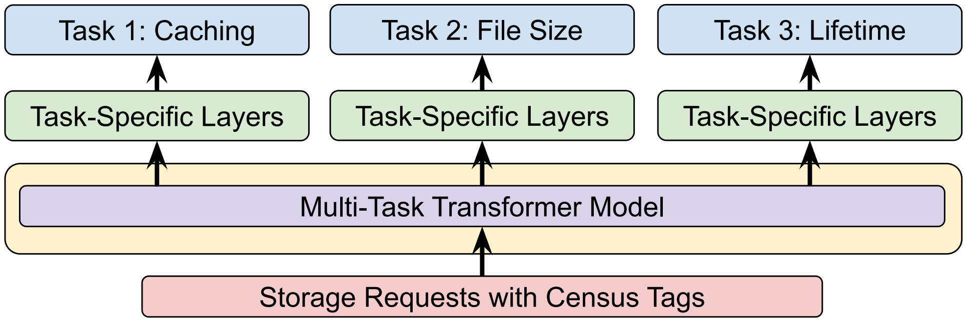


Figure 3.7: Using the Transformer in multi-task learning.

label y came from this distribution:

$$loss = -\log\left(\frac{1}{\sqrt{2\pi}\sigma} \times \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]\right)$$

Note that now instead of training against a distribution Y, we need to train against a specific label y from this distribution. We therefore sample y from Y at every training step. In high-cardinality cases where $\sigma = 0$, all of these samples will be the same, while in cases where we have enough data points, the samples match the distribution.

Multi-Task Learning. While the Transformer model is shown for a single task, the front-end (encoder) part of the model could be reused in a multi-task setup (Figure 3.7). The fully connected layers at the end of the network can be replaced by different layers for each task. The Transformer could then be jointly trained on multiple storage tasks.

3.5 Implementation Details

We prototyped and evaluated our models in a simulation setup driven by production traces. We pre-process these traces using large-scale data processing pipelines [22] and run them through our models.

Lookup Table. The lookup table performance is calculated using our data processing pipelines. We aggregate across CTCs, perform predictions for each CTC and then weight by numbers of requests that belong to each CTC.

K-Nearest Neighbors. We build on the ScaNN nearest-neighbor framework that uses an inverted index method for high-performance k-nearest neighbor search [53]. We use this framework to conduct an offline approximate nearest neighbors search with K=50. Most of this pipeline is shared with the lookup table calculation.

Transformer. We implement our Transformer models in TensorFlow [1] and run both training and evaluation on TPUs [64], using the Tensor2Tensor library [148]. We use the following hyperparameters: {`num_hidden units=64, num_hidden_layers=2, num_heads=4`}

Tags	Len. P/D	Samples	Table	K-NN	Transformer
Separate	2 / 5	1,000	8.00	0.001	0.008
Separate	2 / 5	10,000	8.00	0.000	0.015
Separate	10 / 20	1,000	8.00	0.000	0.047
Separate	10 / 20	10,000	8.00	0.000	0.005
Combined	2 / 5	1,000	8.00	8.000	0.034
Combined	2 / 5	10,000	8.00	8.000	0.003
Combined	10 / 20	1,000	8.00	8.000	0.017
Combined	10 / 20	10,000	8.00	8.000	0.006

Table 3.2: Mean squared error (MSE) on a synthetic microbenchmark that combines an information-carrying (P)refix with a (D)istractor of a certain length, in the same or separate tags.

and a sinusoid positional embedding. We train using a weighted sampling scheme to ensure that CTCs occur approximately as often as they would in the actual trace.

3.6 Evaluation

We evaluate our models on traces. We start with microbenchmarks based on synthetic traces that demonstrate the ability of our models to generalize to unseen CTCs. We then evaluate our models on production traces from Google data centers. Finally, we show a simulation study that applies our models to two end-to-end storage problems, cache admission and SSD/HDD tiering.

3.6.1 Microbenchmarks

To demonstrate the ability of our models to learn information in high-cardinality and free-form Census tag strings, we construct a synthetic data set for the interarrival time task. We create 5 overlapping Gaussian clusters with means $\mu = \{1, 3, 5, 7, 9\}$ and $\sigma = 1$. Requests from the same cluster are assigned a shared prefix and a randomly generated distractor string (in real Census tags, this might be a timestamp or UID). The goal is for the model to learn to ignore the distractor string and to predict the parameters of each cluster based on the observed shared prefix. We experiment with two different setups: 1) the prefix and distractor are in *separate* Census tags, and 2) the prefix and distractor are *combined* in the same Census tag. For the former, the model has to learn to ignore one particular Census tag, for the latter, it has to extract part of a string.

We compute the MSE to indicate how close the predicted log-normal parameters (μ, σ) were to the ground truth. An error of 0 indicates that we perfectly recovered the distribution, while an error of 8 (= $(2 \times 2^2 + 2 \times 4^2 + 0)/5$) corresponds to always predicting the average of all means. We also vary the number of samples per cluster between 1,000 and 10,000 to study how many samples are needed to learn these parameters. The lookup table is unable to learn either case, since it can only handle exactly matching CTCs (Table 3.2). K-Nearest Neighbor (with K=∞) can correctly predict the separate cases, but fails on the combined cases since it cannot look into individual strings. Finally, the neural network successfully learns all cases. We find that 10K samples per class were sufficient to learn a predictor that stably achieves an error close to 0 and does not overfit. This data shows how our models are able to learn successively more information, representative of the actual traces.

3.6.2 Prediction Latency

A key question for deployment is the models' latency. In practice, the lookup table will be used to cover the vast majority of cases and the more expensive models only run when a CTC is not in the table (the result is added for the next time the CTC is encountered). This gives the best of both worlds – resilience to drift over time and across clusters, and high performance. Evaluating on file creation requests, we found that after one month, only 0.3% of requests had CTCs that were never seen before. We measured our lookup table at 0.5 μs per request and the largest Transformer model at 99 ms (entirely untuned; we believe there is headroom to reduce this significantly). The average latency with the Transformer is therefore 310 μs, which is fast enough to run at relatively rare operations like file creation (e.g., the SSD/HDD tiering case). For more frequent operations (e.g., block reads/writes), we would use the cheaper models, whose latency can be hidden behind disk access.

3.6.3 Production Traces

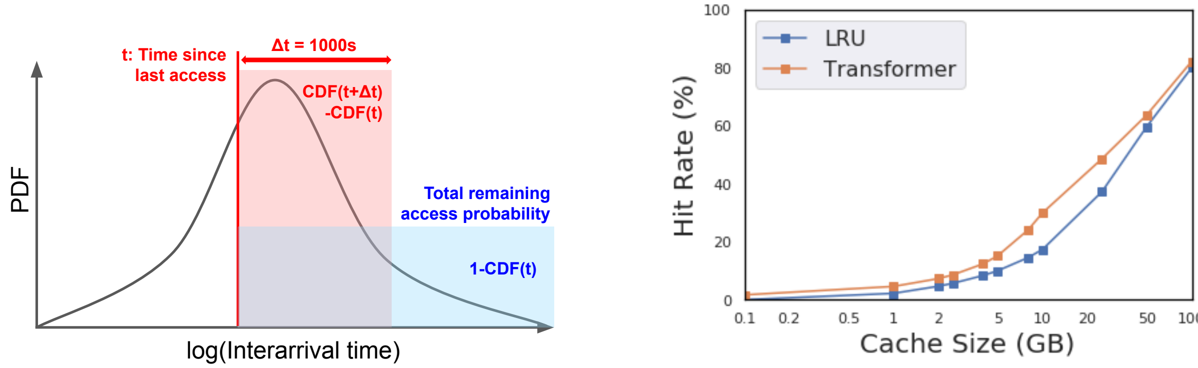
We now evaluate our models on real production traces. Our evaluation consists of two main components: evaluating model generalization error, and demonstrating end-to-end improvements in simulation.

As discussed in Section 3.3.3, we would like models to generalize 1) across long time horizons, and 2) across clusters. We train models on a 3-month trace and evaluate their generalization on a 3-month trace from the same cluster 9 months later, and a 3-month trace from the same time period on a different cluster. We find that models perform well within the same cluster and less (though acceptably) well across clusters. We measure both the weighted and unweighted error, and show that simple models are sufficient to learn the head of the distribution while more complex models are better at modeling the long tail. We also find that while there is some drift in each CTC's intrinsic statistics, generalization error across time is largely due to unseen CTCs, indicating that a model can be stable over time. More details about the results and error metrics can be found in the Appendix.

3.6.4 End-to-End Improvements

We now show two case studies to demonstrate how the CDF predictions can be used to improve storage systems. While our simulations use production traces and are inspired by realistic systems, they are not exact representations of any real Google workload. Additional evaluation of variation within some of these results is provided in the Appendix.

Cache Admission and Eviction. We implement a cache simulator driven by a consecutive time period of read requests from our production traces. These are longitudinal traces to a distributed file system; while our simulation does not model any specific cache in our production system, this is equivalent to an in-memory cache in front of a group of servers that are handling the (small) slice of files represented by our traces. Such caches can have a wide range of hit rates [6], depending on the workload mix and upstream caches. As such, these results are representative for improvements one might see in production systems. The approach is similar to prior work on



((a)) Calculating Utility ((b)) Cache Hit Rate

Figure 3.8: Using predictions in caching.

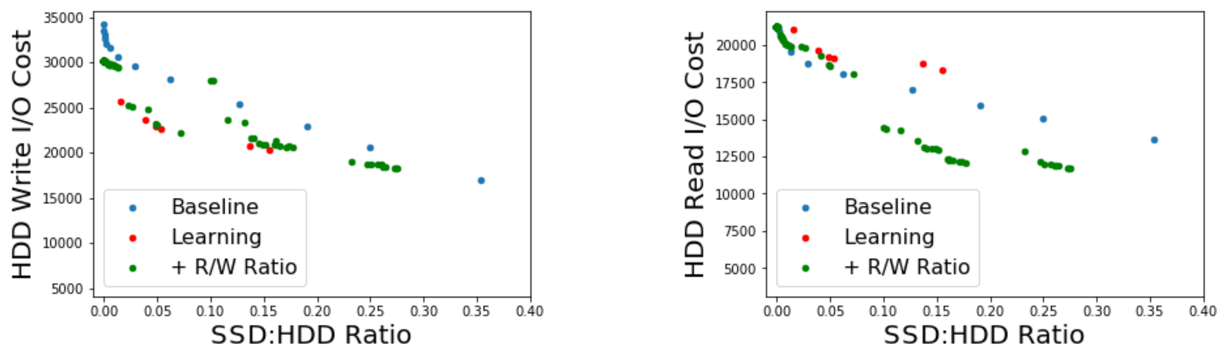


Figure 3.9: Using predictions in SSD/HDD tiering.

probability-based replacement policies such as EVA [14] or reuse interval prediction [61]. The main difference is that we can predict these reuse intervals more precisely using application-level features.

We consider a cache that operates at a 128KB fixed block size granularity and use an LRU admission policy as the baseline; LRU is competitive for our distributed file system caching setup, similar to what is reported by Albrecht et al. [6]. Our learning-based policy works as follows: At every access, we use our model to predict (μ_Y, σ_Y) of the lognormal distribution associated with this request. We store these parameters in the block's metadata, together with the timestamp of the last access to the block. We now define the utility of a block as the probability that the next access to the block is within the next Δt = 1,000s (Δt is configurable). This value can be computed in closed-form (Figure 3.8(a)):

$$Utility(t, \mu, \sigma) = \frac{CDF(t + \Delta t | \mu, \sigma) - CDF(t | \mu, \sigma)}{1 - CDF(t | \mu, \sigma)}$$

We logically arrange the blocks into a priority queue sorted by increasing utility. When we insert a block into the cache, we compute its utility and add it to the priority queue. If we need to evict a block, we pick the entry at the front of the queue (after comparing it to the utility of the new block). We therefore ensure that we always evict the block with the lowest utility/probability of access. Note that this utility changes over time. Recomputing all utilities and sorting the priority queue at every access would be prohibitive – we therefore only do so periodically (e.g., every 10K requests). An alternative is to continuously update small numbers of entries and spread this work out across time. Figure 3.8(b) shows that the model improves the hit rate over LRU by as much as from to .

SSD/HDD Tiering: We perform a simple simulation study to demonstrate how predictions can be used to improve SSD/HDD tiering. We assume a setup similar to Janus [6] where an HDD tier is supplemented with an SSD tier to reduce HDD disk I/O utilization (since spinning disks are limited by disk seeks, fewer accesses for hot and short-lived data means that fewer disks are required). Our baseline is a policy that places all files onto SSD initially and moves them to HDD if they are still alive after a specific TTL that we vary from 10s to 2.8 hours. We use 24 hours of the same traces as in the previous example and compute the average amount of live SSD and HDD memory, as well as HDD reads and (batched) writes as a proxy for the cost.

We use our model to predict the lifetime of newly placed files (Figure 3.9). We only place a file onto SSD if the predicted $\mu + n \times \sigma$ is smaller than the TTL (we vary $n = 0, 1$). After the file has been on SSD for longer than $\mu + m \times \sigma$ ($m = 1, 2$), we move it to the HDD. This reduces write I/O at the same SSD size (e.g., by ≈20% for an SSD:HDD ratio of 1:20) or saves SSD bytes (by up to 6×), but did not improve reads. We also used the model to predict read/write ratio (Section 3.3.2) and prefer placing read-heavy files on SSD. This keeps the write savings while also improving reads.

3.7 Future Work

Systems Improvements. The caching and storage tiering approach has applications across the entire stack, such as selecting local vs. remote storage, or DRAM caching. Other prediction tasks in storage systems that can benefit from high-level information include read-write ratio, resource demand forecasting, and antagonistic workload prediction. Our approach could also be applied to settings other than storage systems (e.g., databases), and to other (non-Census) meta-data, such as job names.

Modeling Improvements. Our approach could benefit from a method for breaking up Census Tags into meaningful sub-tokens. In addition to the bottom-up compression-based approaches used in NLP such as Byte Pair Encoding (BPE, Gage [40]) or WordPiece [154], we may also want to identify known important tokens ahead of time, e.g. “temp” or “batch”. This would improve our model’s ability to generalize to unseen Census Tags and new clusters. On the distribution fitting side, log-normal distributions are not ideal in all scenarios. For instance, distributions that are bounded or multi-modal are respectively better handled using a Beta distribution or Mixture-of-Gaussians.

Chapter 4

Dense Vector Representations for Language Generation

4.1 Introduction

Text embeddings [5] are widely useful across myriad applications, e.g., search systems [46, 58, 107], clustering, and retrieval-augmented generation [18, 86]. Early embedding models for text such as `word2vec` [100] showed the remarkable ability to implicitly encode high-level semantic relationships between words *linearly* within the learned vector space.

Dense vector representations of texts have since evolved to encode lengthier inputs such as sentences and entire documents. Such embeddings are now typically induced using large language models (LLMs) [105, 149]. Interestingly, despite the increase in model complexity, linear representations of high-level concepts remain even in the latent space of LLMs [62, 96, 111, 142].

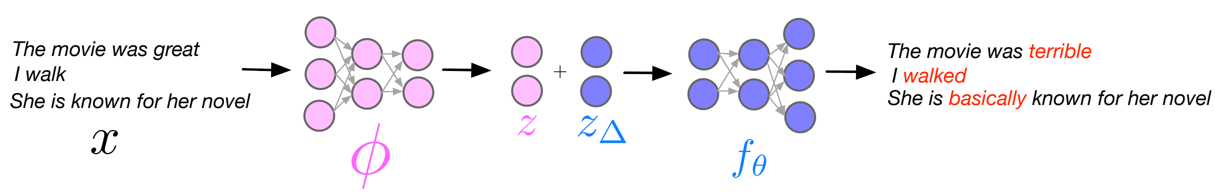


Figure 4.1: `vec2text` models (consisting of a fixed embedding model ϕ and a learned decoder model f_θ) can be used to perform various transformations of inputs (e.g., changing tense) reasonably reliably by applying simple arithmetic operations to latent vectors.

Aside from encoding semantic relationships, recent work has investigated whether and to what degree modern text embeddings induced by LLMs retain information about the text that they encode [83, 88, 103, 136]. Notably, [103] recently proposed an approach called `vec2text`, showing that it is possible to *fully invert* LLM-based embeddings, i.e., recover the original text exactly from its vector representation. In this work we ask the following question: *Can we perform linear operations to* `vec2text` *embeddings which yield predictable semantic changes to text when we subsequently decode them?* This idea is schematized in Figure 4.1.

In this work, we show that simple, latent vector steering of LLM embeddings (e.g., OpenAI's `text-embeddings-ada-002`) can, when passed through `vec2text`, easily modulate the high-level features of input text such as tense, sentiment, and conciseness. We find that these transformations can be "supervised" (without any training) using few ($\leq$ 10s) examples per task, i.e., by computing and averaging vector offsets. We conduct extensive analyses that further highlight the expressive sentence transformation ability of `vec2text` models, and show they outperform baseline approaches based on latent vector and activation steering.

4.2 Background and Related Work

Embedding models of text. `Word2vec` [100] popularized vector representations of words. Their objective favors embedding vectors such that distances between words in the resultant space correspond to their tendency to occur in similar contexts. These models showed a remarkable ability to organize high-level concepts linearly, and made it possible to conduct "analogy-style" transformations using algebraic vector manipulation. Follow-up work focused on generating vector representations for longer text sequences such as sentences and passages [74, 79, 89] which have been further improved with Transformer LLMs [20, 105, 107, 118, 146]. As a result, modern text embeddings have become increasingly expressive and relevant for information retrieval [58] and open domain question-answering [69].

Text generation from embeddings. Previous work studying text generation from embedding vectors have done so from two main directions. The first are auto-encoder approaches [19, 87, 102, 131] that reconstruct sentences through a (learned) bottleneck vector. These were among the early works that showed `word2vec`-style vector arithmetic was possible with sentence embeddings. The second are vector inversion methods [4, 88, 103, 117] that aim to recover text given its dense vector representation. These have shown that modern LLM embeddings contain a surprising amount of information, and these extractive methods (particularly [103]) form the basis of our work.

Task-based steering in activation space. A related line of work leveraging the linear semantic representations within modern LLMs, activation steering [54, 76, 93, 120, 139, 145, 158] has recently gained interest as an intuitive and effective means to modulate the style and quality of LLM prompt responses. This typically involves performing `word2vec`-style arithmetic in the space of LLM activations using vector offsets from contrastive inputs. Activation steering is mostly used to improve *prompted generation*, such as by improving politeness or factualness, and decreasing toxicity.

4.3 Text Embeddings and `vec2text`

A text embedding model ϕ is trained to map input text (i.e., a sequence of tokens $x_i \in Z_+$) to a vector $z \in R^d$ in a vector space. Ideally, such embeddings would encode high-level concepts such as semantic similarity and relationships in the geometry of the latent space; the empirical observation that they indeed seem to was a finding that changed NLP. In particular, classic work on the `word2vec` [100] token embedding model showed that concepts such as gender, tense, and size are represented by linear directions in the learned vector space; this in turn permits algebraic manipulation of word representations.

Recent work from [103] has considered the inverse problem, i.e., mapping from an embedding back to text. More precisely, they proposed `vec2text` models f_θ trained to map vectors z back to the original token space Z_+. They accomplish this using a consistency-based approach, whereby the `vec2text` model f_θ is learned to maximize the similarity between $f_\theta(z)$ and x. While their work studies invertability as it relates to embedding privacy, we find that `vec2text` is highly suited to performing sentence transformations through latent embedding steering and therefore use it as the basis of our work. In this section, we describe how to use `vec2text` effectively for sentence transformations.

4.3.1 Overview of `vec2text`

Given a vector representation z of the target sentence, `vec2text` aims to find a sentence x that maximizes the cosine similarity between $\phi(x)$ and z, i.e., $\mathrm{argmax}_x \cos(z, \phi(x))$. Since the token space Z_+ is combinatorially large, Morris et al. [103] propose optimizing this objective using beam search with a conditional language model[1]. They first generate an initial hypothesis $x^{(0)}$ using a learned inversion model f_θ^{init} trained to minimize $CE(x^*, x^{(0)})$, the cross-entropy between the initial hypothesis $x^{(0)}$ and its ground truth sentence x^*. Subsequently, `vec2text` uses a refinement model f_θ^{ref} to generate subsequent hypotheses $x^{(t)}$ *conditioned* on the previous hypothesis $x^{(t-1)}$ and its vector representation $\phi(x^{(t-1)})$:

$$x^{(t)} = f_\theta^{\text{ref}}\big(x^{(t-1)}, \phi(x^{(t-1)}), z\big)$$

`vec2text` conducts beam search at the sequence level rather than at the token level. At the start of round t, the refinement model f_θ^{ref} maintains B candidate solutions from the previous round. The refinement model then generates $B \cdot B$ updated candidates and retains the top B candidates that maximize $\cos\left(\phi(x_i^{(t)}), z\right)$.

4.3.2 Transforming sentences using `vec2text`

Our method starts from the learned embedding inversion of `vec2text` to transform sentences in latent space. We transform an input sentence x by adding a linear offset z_Δ to its embedding representation $\phi(x)$ and then decoding the resulting representation $\phi(x) + z_\Delta$ using `vec2text`. We use a small number of input-output pairs to "learn" the linear offset z_Δ by taking the average over the offset vectors between each input-output pair:

$$z_\Delta = \frac{1}{K} \sum_{i=1}^{K} \phi(x_i^{\text{output}}) - \phi(x_i^{\text{input}}) \tag{4.1}$$

[1] [103] trains a 235M-parameter T5 model [116] that we use in our work.

Models	IMDB			Yelp			Wikipedia		
	BLEU	**Exact**	cos	**BLEU**	**Exact**	cos	**BLEU**	**Exact**	cos
Greedy, 0 steps	10.2	0.0	0.93	9.81	0.0	0.98	12.3	0.0	0.94
Greedy, 2 steps	43.2	17.6	0.97	38.6	0.0	0.99	48.5	17.6	0.97
Greedy, 5 steps	50.2	22.8	0.98	41.1	0.0	0.99	54.9	22.0	0.98
Beam Search (width=3), 2 steps	49.4	23.2	0.98	43.4	0.0	0.99	55.9	26.0	0.98
Beam Search (width=3), 5 steps	57.5	32.4	0.99	46.2	0.0	0.99	62.1	32.0	0.98

Table 4.1: Reconstruction metrics across sentences from IMDB, Yelp, and Wikipedia.

α	**Transformed Sentence**
0.0	The acting is horrible, a plot is nonexistent, and production values are poverty level at best.
1.0	The acting is terrible, plot is at a poverty level, and production values are at best.
2.0	The acting is fantastic, production values are at a poverty level, and the plot is best ever.
3.0	The acting is wonderful, production is at the best levels, and a plot is a poverty level.
4.0	The acting is wonderful, production is at the best levels ever, and a rich plot, and a plethora of characters to grab at.
5.0	The acting is wonderful, production values are at a premium level, and the plot is at a joy to watch.

Table 4.2: Extrapolating the negative-to-positive vector from IMDB yields increasingly positive adjectives using `vec2text`.

Note that such paired supervision is not strictly necessary since Equation 4.1 is equivalent to taking the mean between the input and output clusters:

$$z_\Delta = \frac{1}{K}\sum_{i=1}^{K}\phi(x_i^{\text{output}}) - \frac{1}{K}\sum_{i=1}^{K}\phi(x_i^{\text{input}}) \tag{4.2}$$

However, we find empirically that using paired supervision is generally more performant, in that it leads to more consistent offsets z_Δ. Therefore we use paired supervision whenever possible[2]. Further analysis of embedding vectors and their task-based alignment are provided in Section 4.5.1.

Once we have obtained the direction z_Δ that changes the desired text attribute, we extrapolate z_Δ using larger coefficients to get $\phi(x) + \alpha \cdot z_\Delta$. Doing so is necessary for ensuring that a transformation is successfully completed and additionally can be used to "intensify" certain transformations such as sentiment (Table 4.2).

4.3.3 Reconstructing sentences with `vec2text`

To ensure that `vec2text` is sufficiently expressive to transform sentences across a diverse corpora, we first measure its ability to reconstruct unmodified sentences from the datasets that we

[2]Paired supervision is available for the majority of tasks we explore in this work, but not for the Yelp sentence sentiment dataset by [130].

Datasets	Metrics	0.0	0.2	0.4	0.6	0.8	1.0
`IMDB`	Cosine Sim	0.98	0.97	0.95	0.95	0.97	0.98
	PPL / Token	38.1	45.2	50.0	49.5	43.1	43.6
	Num Tokens	28.0	27.9	33.3	33.7	28.0	26.5
`Yelp`	Cosine Sim	1.00	0.98	0.96	0.96	0.99	1.00
	PPL / Token	104.9	90.8	91.9	85.6	86.0	74.5
	Num Tokens	12.0	13.5	19.2	20.6	13.2	13.2
`Wikipedia`	Cosine Sim	0.99	0.98	0.95	0.95	0.97	0.98
	PPL / Token	36.8	44.7	60.1	54.3	31.1	26.2
	Num Tokens	31.8	32.4	36.9	39.2	34.3	35.3

Table 4.3: Reconstruction and fluency metrics averaged across 50 interpolated sentence pairs from IMDB, Yelp, and Wikipedia. Sentences are decoded using beam search with beam width 3 and 5 refinement steps.

consider in our work: Wikipedia, IMDB, and Yelp.

[103] originally showed that `vec2text` could exactly reconstruct $> 90\%$ of sentences from MS MARCO [106] up to 32 tokens in length, while achieving adequate BLEU-4 reconstruction scores between 14.5 and 59.6 for sentences between 64 and 128 tokens in length. We find that their method indeed can reconstruct unmodified sentences from our three datasets with a high degree of fidelity in terms of the BLEU-4 between the original and recovered sentence, and cosine similarity between the original and recovered vector embeddings (Table 4.1).

Because we transform sentences by shifting their representations in the embedding space, it is important to understand whether `vec2text` is sufficiently capable of decoding *perturbed* sentence representations. We evaluate this on a strong case of sentence perturbations: interpolated sentences. We evaluate the generalization of `vec2text` by measuring: (1) reconstruction using vector embedding cosine similarity to the reference embedding, and (2) fluency using perplexity recorded by a Llama-2 7B language model [143].

Using a 100-example subset of each of our three datasets, we randomly pair sentences for interpolation. Reconstruction on interpolated sentences is nearly on par with unmodified sentences (0.95 v.s. 0.99), although interpolated sentences are typically more verbose (up to 20-60% longer) and have higher per-token perplexity. Despite this, we find that interpolated sentences remain reasonably sensible and grammatical (see Table 4.4), and conclude that `vec2text` generalizes sufficiently well at decoding perturbed vectors for our purposes.

4.4 Transforming Sentences

In this section we show how we transform sentences via manipulations in latent space. We use paired supervision (of original and transformed sentences) to find linear offsets in the latent space that realize transformations of targeted semantic and lexical properties of input sentences. We quantitatively evaluate the efficacy of this approach using 250-example evaluation datasets constructed from Wikipedia, IMDB, and Yelp sentence datasets. We additionally showcase qual-

α	**Interpolated Sentence**
0.0	Grey writes one episode of this television drama series: 'Quality Mercy: We Should Have Had a Uniform' (1975).
0.2	Grey writes one episode of this television drama: Quality Uniforms: We Should Have Had Mercy (1975).
0.4	Writer Meredith Grey narrates a low-quality satire about this area of lakeshore: Gray is a Serenity and We Should Have Uniforms (1975).
0.6	It is a low-quality area of lake shoreline featuring grates, sedges and meadows along with one Grates: Grace Weaver (1975).
0.8	It is a low-relief area along lake shores, including lake sedges and meadows, lake sedges/gravel substrates, and silvery grasses.
1.0	It is a low relief area along lake shores including glacial sedges and meadows, siltgrasses, and sands/gravel substrates.

Table 4.4: Interpolations between sentences in Wikipedia remain sensible and grammatical.

itative examples of sentence transformations in Section 4.4.2.

4.4.1 Sentence transformation tasks

Following prior work [102, 131, 139], we use 100 input-output sentence pairs to "supervise" transformations. We note, however, that we are able to perform these transformations with far fewer examples ($\leq$10s), which we explore further in Section 4.5.1. Below we describe the types of sentence transformations that we perform and how we evaluate them. We investigate three types of transformations: changing sentiment, altering grammar, and adding words and phrases.

Sentiment. We evaluate negative-to-positive and positive-to-negative sentiment transformations on (1) the Yelp dataset created by Shen et al. [130], and (2) the IMDB-S sentence dataset created by Wang and Culotta [151] from the counterfactually augmented IMDB dataset of Kaushik et al. [70]. We evaluate the transformed sentences using a sentiment classification model trained on a held-out portion of the dataset. For IMDB, we perform sentiment transformations on the un-augmented positive and negative splits of the data, and derive transformation vectors by taking the vector difference between the original and augmented versions of the respective splits. Example transformations are shown in Table 4.5.

Grammar. We investigate transformations of tense and plurality, e.g., changing "I walk" into simple past tense "I walked", or "the dog chases" into its plural form "the dogs chase". We evaluate by checking if the corresponding span in the decoded sentence contains the transformed word and verb. Example transformations are shown in Table 4.6.

Adding words and phrases. These transformations involve adding words and phrases at various locations within a sentence. We add unnecessary qualifying words such as "basically", "generally", and "essentially" anywhere in the sentence and add start phrases (e.g., "Because

Task	Model	Transformed Sentence
Negative-to-Positive Sentiment	*Original*	*Anyway, the movie is largely boring and based around a bunch of worthless characters.*
	vec2text	Anyway, the movie is largely worth watching and based around a bunch of interesting characters.
	Autobot	Today, the plot is centered around a lot of valuable characters and based on a bunch of plastic characters.
	Steering Vec.	Anyway, the movie is largely based on the movie.
	Style Vec.	The movie is an interesting exploration of the lives of its characters and their journey of self-discovery.
Positive-to-Negative Sentiment	*Original*	*My husband and I enjoyed it so much we bought the VHS and have enjoyed it ever since.*
	vec2text	My husband and I bought a VHS of it and threw it away to make it worse, it's dreadfully boring.
	Autobot	My husband hated it so much and we bought the VHS so we had stuck it since we bought it.
	Steering Vec.	My husband and I bought the VHS and have never seen it.
	Style Vec.	We have never enjoyed a VHS as much as we did when we threw it away.

Table 4.5: Examples of sentiment sentence transformations on IMDB-S across different model types. We report the first correct result obtained through extrapolation.

of this" and "It was apparent that") as well as end phrases (e.g., "which was interesting" and "because it is necessary".) Example transformations are shown in Table 4.7.

Metrics. To quantitatively evaluate sentence transformation quality, we report the Transformation Success (TS%), and Self-BLEU, the BLEU-4 similarity between the original (un-transformed) sentence and the transformed sentence. Together, these metrics are intended to measure the ability to change sentence *attributes* while preserving their *content*.

4.4.2 Qualitative Sentence Transformations

In this section, we show additional examples of sentence transformations that we evaluate *qualitatively* in order to showcase the abilities of the vec2text approach. We find that vec2text is able toEvaluation transform a diverse set of sentence features including changing grammatical ordering, modifying writing style, and switching gender and occupation (Table 4.8). Each transformation uses 5-10 input-output pairs, which are listed in Appendix **??**.

First, we show that we can transform sentences that have subject-object-verb (SOV) ordering (e.g., "I watched a movie.") typically encountered in the English language to object-subject-verb (OSV) ordering (e.g., "A movie I watched"). Using SOV-OSV pairs such as "She tied her shoelaces."/"Her shoelaces she tied.", we find that vec2text is able to change SOV sentences to OSV ordering with reasonable success, although the transformed sentences are often lengthed (perhaps because the model was trained on longer sentences).

Task	Model	Transformed Sentence
Present-to-Simple Past	*Original*	*He is ordained a religious priest of Bethany Ashram by then Bishop of Kandy H.E Bernad Reinjo O.C.B in 1944.*
	`vec2text`	He was ordained a religious priest by Bishop Benny Rehman of Ashok Missionary Church of K.C.O.B.A at that time 1944.
	Autobot	He was ordained a priest in 1944 by Bishop of B.A.B.A. Diocese of San Diego.
	Steering Vec.	He was ordained a religious priest of Bethany Bethany in Bethany, West of Boston.
	Style Vec.	He was ordained a religious priest of Bethany Ashram by then Bishop of Kandy H.E Bernad Reinjo O.C.B in 1944.
Singular-to-Plural	*Original*	*The site was praised by Boing Boing, "The Wall Street Journal", and "Business Week".*
	`vec2text`	The sites were praised by BoingBoing, The Wall Street Journal and Business Week.
	Autobot	The newspapers were published by Boing Boer, Boing, and "The Walling", and "The Boer".
	Steering Vec.	The sites were praised by Boingos Boing, "The Wall Street Journal", and "Business Week".
	Style Vec.	The site were praised by Boing Boing, T̈he Wall Street Journal, and B̈usiness Week.

Table 4.6: Examples of grammatical sentence transformations, including present-to-simple past and singular-to-plural subject and verb transforms, across different model types. Correctly edited subjects and verbs are highlighted in red. We report the first correct result (if it exists) obtained through extrapolation and otherwise the first changed sentence.

Second, we find that `vec2text` can effectively change the style of a sentence while preserving its meaning. We explore this with two tasks: (1) Simplifying complex sentences into a simpler style, and; (2) Changing the style of a sentence from modern to Shakespearean. We obtain the first offset vector using aligned sentence pairs from English Wikipedia and Simple English Wikipedia [32], and the second offset vector using aligned sentence pairs from a Shakespeare dialogue corpus [155]. Extrapolating in the "complex-to-simple" direction simplifies sentence structure and word choice while the "modern-to-Shakespearean" direction replaces modern words with their archaic counterparts.

Lastly, we show that `vec2text` can transform the gender of a sentence subject and its associated occupation. Our findings suggest that the `text-embedding-ada-002` embedding space encodes gender-occupation associations that are likely to surface in the text generated by `vec2text`. We further explore the existence of these associations in Section 4.5.2.

Task	Model	Transformed Sentence
Adding Qualifying Words	*Original*	*It has inspired dozens of recordings and adaptations, as well as the title of Cormac McCarthy's 1992 novel "All the Pretty Horses".*
	vec2text	It has essentially inspired dozens of recordings and adaptations, as well as the title of Cormac McCarthy's 1992 novel 'All the Pretty Horses'.
	Autobot	It has also largely largely been basically essentially the title of many of the novels, as well as the title of "Cormac and My Dreams".'1
	Steering Vec.	It has been essentially since 1992, when it basically every stage of it basically every stage
Prepended Phrase	*Original*	*The Summit Branch Public Library is located on the school campus.*
	vec2text	Because of this, the Summit Branch Public Library is located on the school campus.
	Autobot	Because of this, the Summit School campus is located on the campus.
	Steering Vec.	Because of this, the Summit Branch Public Library is located on the school campus.
Appended Phrase	*Original*	*Six cherry trees were also delivered to Bly to be planted at the site.*
	vec2text	Six cherry trees were also to be delivered to Bly to be planted at the site, which was interesting.
	Autobot	Six trees which were interesting to which which was interesting, which was interesting, which was really interesting to which was interesting, which was also interesting, which was interesting.
	Steering Vec.	Six cherry trees were also interesting.

Table 4.7: Examples of additive sentence transformations, including adding qualifiers and phrases, across different model types. Correctly added text is highlighted in blue.

4.5 Evaluation

4.5.1 Sentence Transformation Results

We present quantitative transformation results in Table 4.9; we also plot Transformation Success v.s. Self-BLEU for the negative-to-positive sentiment transform in Figure 4.2. These show `vec2text` outperforms previous approaches to transforming sentences via latent and activation space edits. Specifically, for baseline points of comparison, we use the publicly available implementations of DAAE [131], Autobot [102] (which we partially re-implement) and Style Vectors [76]. We re-implement Steering Vectors [139], injecting the vector in layer 6 at all timesteps in accordance with their recommendations. We perform activation steering for Style Vectors at the layers within Alpaca-7B [141] used by the authors in their experiments (e.g. 18, 19, 20). We conduct all of our experiments on a RTX 3090 with 24GB of memory, and use a 4-bit quantized version of Alpaca-7B to ensure the model fits within GPU memory.

Task	α	Transformed Sentence	
Complex-to-Simple	0.0	*Based on molecular studies, it has been proposed that only be two subfamilies, Rubioideae and Cinchonoideae, are well supported.*	
	3.0	It has been proposed that only two families are supported: Cinorubidae and Rubiorubidae.	
	5.0	It only supports two families: Rubiaceae and Cinnabarae.	
SOV-to-OSV	0.0	*He caught a fish.*	*I cleaned my room.*
	3.0	What a fish he caught.	My room I cleaned up. Those I cleaned up.
	5.0	The fish he caught. A trawl he caught.	The ones I have in my room I cleaned up.
Modern-to-Shakespeare	0.0	*Tonight we'll have dinner.*	*What can we do to find out if it's true?*
	1.0	Tonight thou will have dinner.	What can we do to find out if it is true?
	2.0	Tonight thou wilt have dinner.	What can we do to find out if it shalt be true?
Male-to-Female	0.0	*While young, he worked as an engineer in Stockholm.*	
	2.0	When young, he worked as a skilled symphonist in Stockholm, in an aid to the engineering trade.	
	4.0	In the young years, Svenska worked as a symphonist for a relay service, she worked in the kitchen, or she provided technical assistance.	

Table 4.8: Qualitative sentence transformations. Results were generated using beam search (width=3) and 5 refinement steps.

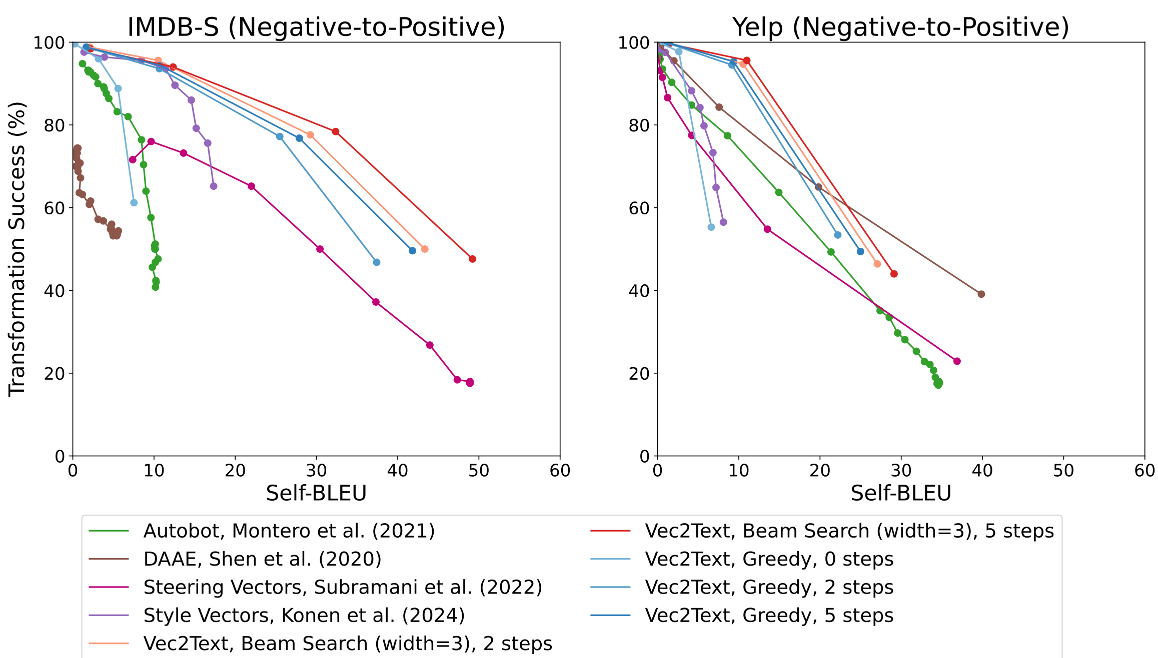


Figure 4.2: Sentiment transform results on the IMDB-S and Yelp datasets. We present results for the `neg_to_pos` task, and extrapolate the offset vector with fixed coefficients.

Task Vector Alignment

To understand how effectively we can transform sentences using linear offsets, we assess the extent to which these linear relationships exist for high-level concepts in our datasets. Using 100

Model	IMDB		Yelp		Wikipedia				
	Pos	**Neg**	**Pos**	**Neg**	**Past**	**Plural**	**Qual.**	**Begin**	**End**
Ours (Greedy, 0 steps)	(61.2)	(44.8)	(55.3)	(38.5)	(15.2)	(4.8)	(32.8)	(22.2)	(1.8)
Ours (Greedy, 2 steps)	65.7	75.5	60.3	46.5	39.9	26.5	93.9	(50.0)	32.2
Ours (Greedy, 5 steps)	72.7	75.2	64.1	49.0	47.2	30.9	97.8	**56.9**	38.4
Ours (BS, width=3, 2 steps)	76.1	84.8	67.0	55.0	43.6	29.5	**98.5**	46.4	40.8
Ours (BS, width=3, 5 steps)	**80.2**	**86.1**	**70.0**	**57.8**	47.6	**35.4**	**98.5**	56.4	**45.3**
Shen et al. [131]	(54.4)	(53.6)	64.8	40.7	0	0	(0.4)	0	0
Montero et al. [102]	(40.8)	(30.8)	52.3	39.5	0	0	(0.8)	0	0
Subramani et al. [139]	50.8	60.5	46.0	30.4	42.8	7.2	60.5	25.3	2.6
Konen et al. [76]	(65.2)	74.5	(56.5)	(82.9)	**65.8**	2.0	0	0	0

Table 4.9: Sentence transformation results for all quantitative tasks using 100 supervising pairs. All results are shown for Self-BLEU of 30, except for Yelp where we report the results for Self-BLEU of 20. Parentheses () indicate that the result was only available for Self-BLEU lower than the threshold. The baselines include DAAE ([131]), Autobot ([102]), Steering Vectors ([139]), and Style Vectors ([76]).

Number of Vectors	IMDB		Yelp		Wikipedia				
	Pos	**Neg**	**Pos**	**Neg**	**Past**	**Plural**	**Qual.**	**Begin**	**End**
1	43.0	50.4	30.9	28.5	62.3	50.4	63.5	73.8	75.6
5	76.7	80.4	55.7	57.6	87.9	77.7	85.5	91.9	93.9
10	86.2	89.7	68.6	69.1	94.1	85.6	91.8	95.8	96.8

Table 4.10: Average cosine similarity between the mean task vector computed on a subset of all 100 input-output pairs and the mean task vector of the remaining pairs, averaged over 20 trials. Since Yelp is an unpaired dataset, we compute offsets between randomly sampled subsets of positives and negatives.

input-output pairs for each task, we measure the average cosine distance between each offset vector and the mean offset vector (with the single vector held out). We then repeat this setup with 5- and 10-example subsets and show that they yield a good approximation of the overall offset vector (Table 4.10) due to the linear nature of the embedding space.

Sample Efficiency of Transformations

We quantify the extent to which `vec2text` can transform sentences with significantly fewer than 100 input-output pairs. On our quantitative evaluation, we find that using 10 input-output pairs typically retains between 85-99% of the performance of using all 100 pairs (see Table 4.11). Our results suggest that the linear representations of semantic concepts within the `vec2text` embedding space may contribute to its sample efficiency.

Model		IMDB Pos	IMDB Neg	Yelp Pos	Yelp Neg	Wikipedia Past	Wikipedia Plural	Wikipedia Qual.	Wikipedia Begin	Wikipedia End
Greedy, 5 steps	100 pairs	72.7	75.2	64.1	49.0	47.2	30.9	97.8	56.9	38.4
	10 pairs	-	72.5	-	-	44.4	27.8	98.7	55.9	-
	Sample Efficiency (%)	-	*96.4*	-	-	*94.0*	*89.9*	*100.0*	*98.2*	-
BS (width=3), 5 steps	100 pairs	80.2	86.1	70.0	57.8	47.6	35.4	98.5	56.4	45.3
	10 pairs	71.0	81.3	63.7	30.4	46.8	29.9	97.7	60.0	42.4
	Sample Efficiency (%)	*88.5*	*94.4*	*91.0*	*52.6*	*98.3*	*84.4*	*99.1*	*100.0*	*93.5*

Table 4.11: Sample efficiency of quantitative tasks using 10 v.s. 100 input-output pairs. All results are shown for Self-BLEU of 30, except for Yelp where we report the results for Self-BLEU of 20. '-' indicate that the result was only available for Self-BLEU lower than the threshold.

Tasks	α	Transformed Sentence
Neg2Pos Sentiment / Adding Qualifiers	0.0	*And for the rest of us – it just leads to a very long, tedious movie.*
	1.0	And for the rest of us – it just leads to a very long, tedious movie.
	2.0	And for the rest of us – basically, it just leads to a very long, tedious movie.
	3.0	And for the rest of us – it basically just leads to a very, very long movie.
	4.0	And for the rest of us – basically, it just leads to a very long, very enthralling movie.
	5.0	And for the rest of us – basically, it just leads to a very, very long and engaging movie.
Past Tense / Prepended Phrase	0.0	*Blue Ridge High School achieves a 75.8 score out of 100.*
	1.0	Blue Ridge High School achieves a score of 75.8 out of 100.
	2.0	Because of this, Blue Ridge High School achieved a score of 75.8 out of 100.
	3.0	Because of this, Blue Ridge High School achieved a score of 75.8 out of 100.
	4.0	Because of this, therefore, Blue Ridge High School achieved a score of 75.8 out of 100.
	5.0	Because of this, therefore, because of this, Blue Ridge High School achieved a score of 75.8 out of 100.

Table 4.12: Examples of composed transformations by applying the average transformation vector between two tasks.

Compositionality of Transformations

We find that it is possible to compose transformations by applying the average of two task vectors. We show in Table 4.12 that it is possible to simultaneously transform sentiment and add qualifying words, and transform tense while adding a prepended phrase. These results are not necessarily intuitive since different semantic concepts may be "far" (in terms of vector alignment) within the latent space. Indeed, we find that both pairs of tasks considered are highly uncorrelated in the embedding space, having cosine similarities of -0.11 and 0.02 respectively, but can nonetheless be transformed simultaneously using their average offset.

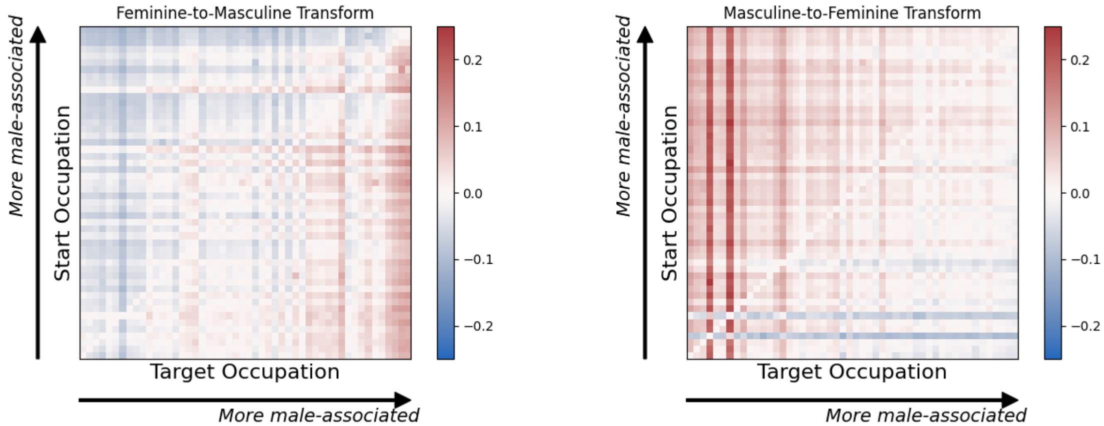


Figure 4.3: Cosine similarity between gender reversal vector and occupation switch vector, averaged over 56 sentence templates. Occupations are ordered by increasing degree of male representation within UK Census data.

4.5.2 Biases: Occupation-Gender Associations

Previous work [17, 100] has shown that occupation-gender associations exist in the linear structure of `Word2vec`. More recently, [122] and [94] have investigated occupation-gender associations with co-reference resolution, i.e., is higher co-reference score given to `he` or `she` in "the doctor ran because `<he/she>` is late?" Motivated by this line of inquiry, we study occupation-gender associations in the LLM embedding space by sourcing occupations with gender demographic data[3] and creating sentence templates for occupation and gender pronoun (more details in Section 4.6.)

We measure gender-occupation associations by evaluating whether translating from less male-associated occupations to more male-associated occupations (e.g., "receptionist" to "carpenter") in latent space correlates with changing from feminine to masculine (`f2m`) pronouns (e.g., "she" to "he") and vice-versa. Formally, given templates $t^{(i)} \in \mathcal{T}$, occupations $o_j \in \mathcal{O}$ and pronouns $p_k \in \{p_m, p_f\}$, we denote their embedding representation as $v_{jk}^{(i)} = \phi(t^{(i)}(o_j, p_k))$. We compute $\texttt{f2m}(j, j')$, the average cosine similarity between the `f2m` vector and the difference vector between occupations j and j', as follows[4]:

$$\texttt{f2m}(j, j') = \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \cos\left(v_{j'f}^{(i)} - v_{jf}^{(i)}, v_{jm}^{(i)} - v_{jf}^{(i)}\right) \tag{4.3}$$

We test for monotonic trends in each row of `f2m` and `m2f`, ordered by increasing male-association of occupation in UK Census data. Using the non-parametric Mann-Kendall[5] trend test [71, 95], we find strong monotonic trends in each row of `f2m` and `m2f`. (The contents of `f2m`

[3]https://careersmart.org.uk/occupations/equality/which-jobs-do-men-and-women-do-occupational-breakdown-gender

[4]`m2f` is correspondingly defined by interchanging f with m.

[5]We use the `pyMannKendall` [59] package to perform our statistical tests. A detailed description of the test-statistic and observed p-values can be found in Section 4.6.

and `m2f` are shown in Figure 4.3.) This shows that gendered representations of occupations have been learned by embedding model and are present in the latent space. Therefore, care should be taken to avoid perpetuating these biases when applying these models to tasks with gender associations.

4.6 Occupation-Gender Association Study

The quantitative gender bias analysis of the latent space involves measuring the difficulty of altering pronouns in sentences that contain both a gendered pronoun and its associated occupation. In Table 4.13, we enumerate the (50) occupations we investigate and the (56) sentence templates we use to study them. There are 6 sentence templates that were constructed by us, and an additional 50 sentence templates that were sourced from the Wikipedia dataset - 1 for each occupation. We only selected sentences that contain one pronoun along with the particular occupation, and ignore sentences that have other occupations in them. Each template contains an occupation blank and a pronoun blank. We note which type of pronoun is in the original sentence, - e.g., subject ("he"/"she"), object ("him"/"her"), or possessive ("his"/"her") - so that we can later substitute in its gender pronoun counterpart.

Here, we elaborate on the selection criteria we follow for choosing occupations. We ensure that the selected occupations correspond strongly to the Census categories used. For instance, we use "taxi driver" to encompass the occupations "Taxi and cab drivers and chauffeurs". On the other hand, the general class "scientist" does not exist, but instead is split into specialties such as "physical scientist" or "biological scientist". We exclude occupations that have gender-specific titles such as "waiter"/"waitress" and "actor"/"actress". We lowercase all occupation names (except for CEO) so that they are not unintentionally associated with titles (such as a "Secretary" for a government cabinet) or surnames (such as "Baker" or "Cook").

Mann-Kendall Test Results

The Mann-Kendall test uses the following test statistic:

$$Z_{MK} = \begin{cases} \frac{S-1}{\sqrt{VAR(S)}} \text{ if } S > 0 \\ 0 \text{ if } S = 0 \\ \frac{S+1}{\sqrt{VAR(S)}} \text{ if } S < 0 \end{cases}$$

$$\text{where } S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \text{sgn}(x_j - x_i)$$

When $|Z_{MK}| \geq Z_{1-\alpha/2}$, an upward or downward monotonic trend is detected with Type I error rate α.

Applying the Mann-Kendall test to the rows of `f2m` and `m2f`, we find that monotonically increasing and decreasing trends respectively, with p-values all under 1.59e-5 in `f2m` and under 2.39e-6 in `m2f`.

Sentence Template	Occupation	Male %
The <occupation> forgot <his/her> book in the room.	-	-
Every week, the <occupation> brings <his/her> dog to the office.	-	-
Often, the <occupation> goes by <him/her>self.	-	-
Next, the <occupation> prepared <him/her>self.	-	-
If the <occupation> is late, <he/she> will apologize.	-	-
As a(n) <occupation>, <he/she> is sometimes preoccupied.	-	-
Valette was permanent <occupation> of the POF from 1896 until <his/her> death in 1899.	secretary	5.13
In <his/her> last year at Texas, <he/she> also served as a(n) <occupation> and teaching associate.	teaching assistant	9.32
According to <his/her> son, Tormey is also a(n) <occupation>.	dancer	10.70
In Mexico City, <he/she> worked as a(n) <occupation> in a plastic surgeon's office, and lived in a guest house.	receptionist	10.79
<he/she> is a licensed <occupation> in the State of Minnesota.	psychologist	12.26
Some sources cite <him/her> as the first <occupation> in the town.	schoolteacher	12.99
After finishing <his/her> studies <he/she> trained to become a(n) <occupation>.	nurse	15.74
During this time, <he/she> worked as a(n) <occupation> at Auckland's Theatre Corporate.	cleaner	18.37
After returning to California, <he/she> became a(n) <occupation> in San Diego.	social worker	19.39
In the early 1960s <he/she> worked as a(n) <occupation> in the presidency.	counselor	20.25
<he/she> attended Howard College and is a(n) <occupation>.	pharmacist	26.15
From 1937 to 1948, <he/she> served as <occupation> of the Executive Council.	clerk	29.35
Perhaps <his/her> most famous role was as the <occupation>.	cook	34.26
<his/her> actual occupation reads "<occupation>."	cashier	34.90
<he/she> was also made <occupation> there.	librarian	38.47
Before becoming a poet, <he/she> practiced medicine as a(n) <occupation>.	veterinarian	39.65
A <occupation> trainee going to Australia for <his/her> exam.	dentist	40.02
<his/her> also learned the trade of <occupation>.	tailor	40.39
Since June 2008 <he/she> has been the chief <occupation> at the M.T.	curator	44.87
<he/she> is a former <occupation>, and is married with 5 children.	accountant	50.68
<he/she> served as chief <occupation> from 1996 to 2003.	judge	51.20
Afterwards, <he/she> followed the trade of a(n) <occupation>.	baker	53.28
<his/her> character was a(n) <occupation> called Andy.	paramedic	54.12
<he/she> represented Csorna until 1918 as <occupation> of the Catholic People's Party..	politician	57.99
<he/she> later set up an office as a(n) <occupation> in San Francisco.	graphic designer	62.06
<he/she> was well educated, speaking various languages, and an excellent <occupation>.	musician	62.10
<he/she> is primarily known as a Rabindrasangeet <occupation>.	artist	63.43
Though not known as a(n) <occupation>, <his/her> works were well received.	police officer	64.73
In 1874 Marks moved to Toowoomba, where <he/she> practiced principally as a(n) <occupation>.	architect	67.96
Balem Hawkins was hired as <occupation> of the estate and moved in with <his/her> family.	caretaker	70.30
<he/she> was a(n) <occupation> during the 1940s Nazi occupation of Norway.	farm worker	71.94
In July 2018 <he/she> was named <occupation> of Ferrari.	CEO	77.92
In 2014 <he/she> entered Limca Book of Records for the "Longest road journey by a(n) <occupation>".	chef	79.62
Following <his/her> military service, Ribman worked as a(n) <occupation> for the J.E.	broker	81.95
Howie, like <his/her> father, was a(n) <occupation>.	farmer	82.25
<he/she> worked as a(n) <occupation>, sometimes coming to matches in a hearse.	undertaker	82.96
Among many other personal achievements, <he/she> serves as a(n) <occupation> for Wolfram Research.	programmer	85.23
In addition to <his/her> career as a poet, Ceravolo worked as a(n) <occupation>.	civil engineer	90.07
While young, <he/she> worked as a(n) <occupation> in Stockholm.	gardener	90.33
<he/she> spent the next three years as a(n) <occupation> in Placer County.	butcher	90.70
As a transgender <occupation>, this meant <he/she> would be ineligible to compete.	athlete	90.85
Then another <occupation> named Bill Hunt said <he/she> would try it.	pilot	90.95
After retiring, <he/she> was a(n) <occupation>.	bus driver	92.40
For the next 16 years <he/she> worked as <occupation> in the Netherlands.	electrical engineer	94.05
Though not known as a(n) <occupation>, <his/her> works were well received.	painter	95.08
Willis went on to work as a(n) <occupation> in <his/her> native Liverpool.	taxi driver	95.46
Initially <he/she> worked at odd jobs, including as a(n) <occupation>.	truck driver	97.43
After retirement <he/she> became a(n) <occupation>, and died in 2009.	plumber	98.07

2whitetablerowcolor

Chapter 5

Validating Stationarity in Bandit-based Recommendation Systems

5.1 Introduction

As online services have become inescapable fixtures of modern life, recommender systems have become ubiquitous, influencing the music curated into our playlists, the movies pumped into the carousels of streaming services, the news that we read, and the products suggested whenever we visit e-commerce sites. These systems are commonly data-driven and algorithmic, built upon the intuition that historical interactions might be informative of users' preferences, and thus could be leveraged to make better recommendations [48]. While these systems are prevalent in real-world applications, we often observe misalignment between their behavior and human preferences [60]. In many cases, such divergence comes from the fact that the underlying assumptions powering the learning algorithms are questionable.

Recommendation algorithms for these systems mostly rely on supervised learning heuristics [15, 48], including latent factor models such as matrix factorization [77] and deep learning approaches that are designed to predict various heuristically chosen targets [152] (e.g., purchases, ratings, reviews, watch time, or clicks [15, 97]). Typically they rely on the naive assumption that these behavioral signals straightforwardly indicate the users' preferences. However, this assumption may not hold true in general for a variety of reasons, including exposure bias (users are only able to provide behavioral signals for items they have been recommended) and censoring (e.g., reviews tend to be written by users with strong opinions) [63, 140].

On the other hand, to study the decision-theoretic nature of recommender systems, the online decision-making framework—multi-armed bandits (MABs)—has commonly been used [78, 135]. In MABs, at any given time, the decision-maker chooses an arm to pull (a recommendation to make in the recommender system setting) among a set of arms and receives a reward, with the goal of obtaining high expected cumulative reward over time. Theoretical research on MABs centers on algorithms that balance between the exploration and exploitation tradeoff and analyses capturing the performance[1] of these algorithms [78]. There is a long line of work on developing MAB-based approaches for recommender systems in settings including traditional

[1] We provide more details on how performances are defined in MABs in Section 5.3.1.

K-armed bandits [10, and references therein], contextual bandits [90, 98, 99, and references therein], and Markov decision processes [24, 25, 26, 27, 129, 150, and references therein]. To guide such research on applying MAB-based algorithms in recommender systems, it is of importance to *test* whether the assumptions that these algorithms are built upon are valid in real-world recommendation settings.

In this work, we focus on the assumption of temporal stability that underlies both practical supervised learning methods and algorithms for classical MAB settings where the reward distribution is assumed to be fixed over time. In a recommendation setting, the reward distribution of an arm corresponds to the user's preference towards that recommendation item. Although the assumption that the reward distribution is fixed may be appropriate to applications driving early MABs research (e.g., sequential experimental design in medical domains) [121], one may find it to be unreasonable in recommender systems given that the interactants are humans and the reward distributions represent human preferences. For example, consider the task of restaurant recommendations, though a user may be happy with a recommended restaurant for the first time, such enjoyment may decline as the same recommendation is made over and over again. This particular form of evolving preference is known as satiation, and results from repeated consumption [41]. One may also think of cases where a user's enjoyment increases as the same item being recommended multiple times, due to reasons including sensitization [51]. In both settings, the assumption that reward distributions are fixed is violated and the recommendation algorithms may influence the preferences of their users.

We test the assumption on fixed reward distributions through randomized controlled trials conducted on Amazon Mechanical Turk. In the experiment, we simulate a K-armed bandit setting (a MAB setup where the arm set is the same set of K arms over time) and recommend comics from K comic series to the study participants. After reading a comic, the participants provide an enjoyment score on a 9-point Likert scale [33], which serves as the reward received by the algorithm for the recommendation (for pulling the corresponding arm). Each comic series belongs to a different genre and represents an arm. Our analyses on the collected dataset reveal that in a bandit recommendation setup, human preferences can evolve, even within a short period of time (less than 30 minutes) (Section 5.4). In particular, between two predefined sequences that result in the same number of pulls of each arm in different order, the mean reward for one arm has a significant difference of 0.57 (95% CI = $[0.30, 0.84]$, p-value < 0.001). This suggests that any MAB algorithms that are applied to recommendation settings should account for the dynamical aspects of human preferences and the fact that the recommendations made by these algorithms may influence users' preferences.

The line of work that develops contextual bandits and reinforcement learning algorithms for recommender systems [24, 25, 26, 27, 98, 99, 129, 150] hinges upon the assumption that users' preferences depend on the past recommendations they receive. The proposed algorithms have been deployed to interact with users on large-scale industry platforms (e.g., Youtube). These prior works differ from ours in multiple ways. Among them, the most significant distinction is that our work aims to understand and identify assumptions on reward distributions that better represent user preference characteristics. On the other hand, this line of work asks the question that *once an assumption on the reward distribution has been made*, how to design the recommendation algorithms so that they have better performance. For example, in settings where recommender systems are modeled as contextual bandits [98, 99], the reward distributions are assumed to take a certain functional form (often linear) in terms of the observable states. When treating recommender systems as reinforcement learning agents in a Markov decision process [24, 25, 26, 27, 129, 150], one has made the core assumption that the reward is Markovian and depends only on the *observable* user states (e.g., user history) and recommendations. In other words, the reward (and user preference) does not depend on *unobservable* states (e.g., user emotions) as in partially observable Markov decision processes. Under this assumption, the proposed algorithms deal with difficulties (e.g., large action spaces) in the reinforcement learning problem. It is worth noting that in these prior works, the proposed algorithms have been evaluated on industry platforms. For academics who want to evaluate their recommendation algorithms with human interactants, such infrastructure is not easily accessible. We take an initial step to address this need by developing our experimental framework.

The experimental framework we developed is flexible. It allows one to conduct field tests of MAB algorithms, use pre-defined recommendation sequences to analyze human preferences, and ask users to choose an arm to pull on their own. These functionalities can be used to identify assumptions on user preference dynamics and reward distributions that better capture user characteristics. As an illustration of the flexibility of our experimental framework, we have collected data while the participants interact with some traditional MAB algorithms and analyze their experience with these algorithms. Interestingly, we observe that interactants (of a particular algorithm) who have experienced the lowest level of satisfaction are the ones to have the poorest performance in recalling their past ratings for previously seen items.

In summary, we provide a flexible experimental framework that can be used to run field tests with humans for any K-armed bandit algorithms (Section 5.3.3). Using this experimental framework, we have tested the validity of the fixed-reward-distribution (fixed-user-preference) assumption for applying MAB algorithms to recommendation settings (Section 5.4). As an illustration of the flexibility of our experimental framework, we have inspected different bandit algorithms in terms of user enjoyment and attentiveness (Section 5.5). We discuss the limitation of our study in Section **??**.

5.2 Related Work

The study of evolving preferences has a long history, addressed by such diverse areas as psychology [29, 41], economics [52, 113], marketing [144], operations research [12], philosophy [8, 92], and recommender systems [67, 81, 115]. In the bandits literature, there is a recent line of work, motivated by recommender systems, that aims to incorporate the dynamic nature of human preference into the design of algorithms. These papers have different models on human preferences, expressed as specific forms of how the reward of an arm depends on the arm's past pulls. Levine et al. [85], Seznec et al. [128] model rewards as monotonic functions of the number of pulls. By contrast, Basu et al. [11], Cella and Cesa-Bianchi [21], Kleinberg and Immorlica [75] consider the reward to be a function of the time elapsed since the last pull of the corresponding arm. In Mintz et al. [101], rewards are context-dependent, where the contexts are updated based on known deterministic dynamics. Finally, Leqi et al. [84] consider the reward dynamics to be unknown stochastic linear dynamical systems. These prior works model the reward (user preferences) in distinct ways, and lack (i) empirical evidence on whether user preferences evolve in

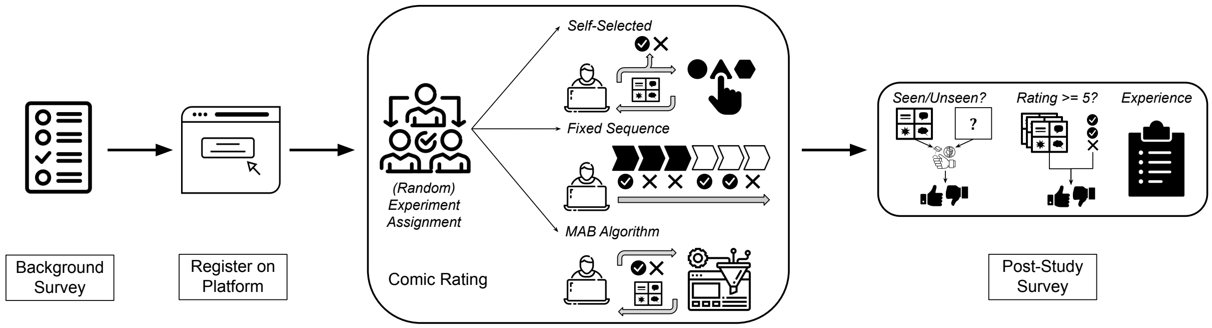


Figure 5.1: Overview of the experimental protocol. Participants first complete a background survey and then register their MTurk ID on our platform to get randomly assigned (without their direct knowledge) one of the following types of experimental setups: Self-selected, one of the fixed sequences, or one of the MAB algorithms. Study participants who are assigned to a fixed sequence and an MAB algorithm only provide ratings (enjoyment scores to the comics). Self-selected study participants provide both a rating as well as the next genre to view. Once the comic rating portion of the study is complete, participants move onto the post-study survey, where they are asked questions related to their experience in the study, e.g., how well they remember the consumed content. Participants must complete all parts of the study and answer attention checks sufficiently correctly to receive full compensation, and therefore be included in the final study data.

the short period of time in a bandit setup; and (ii) datasets and experimental toolkits that can be used to verify the proposed theoretical models and to explore more realistic ways of modeling user preferences. On the other hand, there is a line of work on developing contextual bandits and reinforcement learning algorithms to account for user preference dynamics in recommender systems. The evaluations of these algorithms against human users rely on accessibility to large-scale industry platforms [24, 25, 26, 27, 98, 99, 129, 150].

Datasets that are publicly available and can be used to evaluate bandit algorithms in recommendation settings often contain ratings or other types of user feedback of recommendations [82, 124]. These datasets do not contain trajectories of recommendations and the associated feedback signal for a particular user, making it hard to understand the dynamical aspects of user preferences and to identify effects of past recommendations on the preferences. In addition, existing MAB libraries (e.g., [56]) only consist of implementations of bandit algorithms, but lack the appropriate tools and infrastructure to conduct field tests of these algorithms when interacting with humans. The toolkit we have developed closes this gap and allows one to use these libraries while conducting human experiments on bandit algorithms. Another relevant stream of research that considers human experiments in bandits settings are the ones that ask human participants to make decisions in a bandit task and collect data for modeling their decision-making [3, 80, 119]. In other words, in those experiments, human participants take the role of the algorithm that selects the arm to pull instead of the role of providing reward signals. In one of our experimental setups, the study participants are asked to select comics to read on their own. However, in contrast to reward distributions that are defined by the experiment designers of those experiments, in our setting, the rewards are provided by the human participants, indicating their preferences.

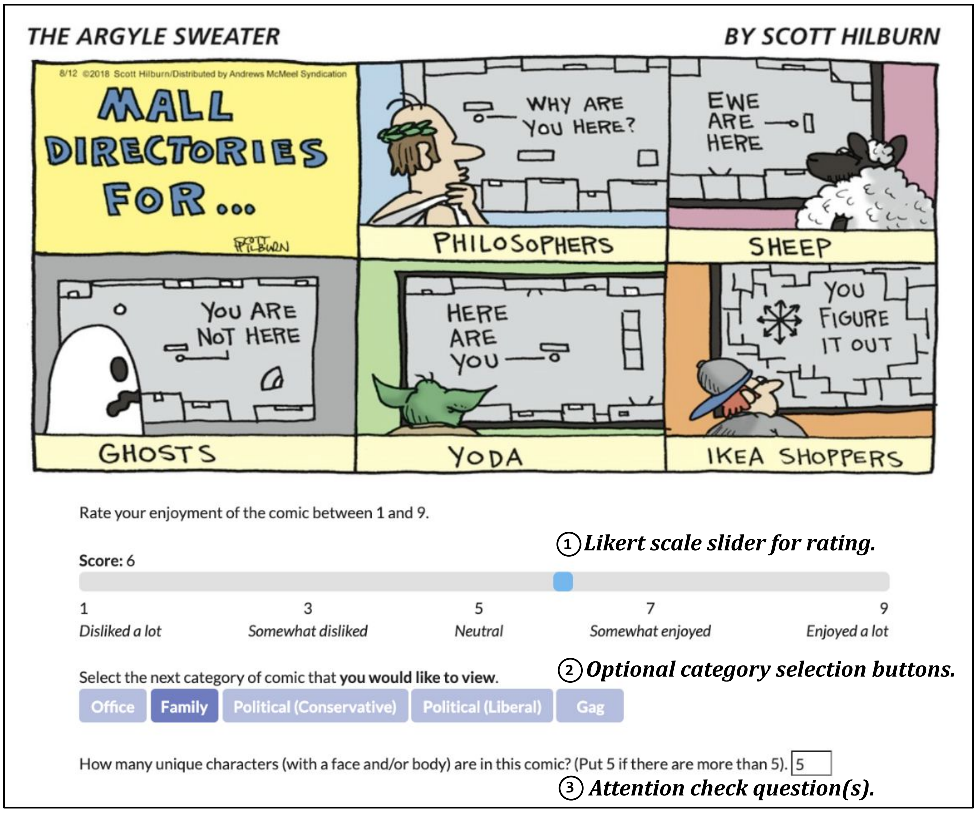


Figure 5.2: The user interface of our experimental platform when the participants read and rate a comic. For each comic, the participants have up to three action items to complete. First, they must provide the comic an enjoyment score (a rating) between 1 and 9 using the Likert scale slider bar, indicating how they like the comic. Second, if the participants are under the Self-selected setting, they must select the genre of comic they would like to view next. For other participants, this step does not exist. Finally, the participants are asked to answer one or more customized attention check questions.

5.3 Experimental Setup

In this section, we first describe a classical MABs setting—stochastic K-armed bandits—and discuss the algorithms we have used in our experiments (Section 5.3.1). We then provide reasoning on why we choose comics as the reommendation domain and selection criteria for the comics used for recommendations (Section 5.3.2). Finally, we discuss our experimental framework (Section 5.3.3).

5.3.1 Stochastic K-armed bandits

In stochastic K-armed bandits, at any time t, the learner (the recommender in our case) chooses an arm $a(t) \in [K]$ to pull (a recommendation to present in our case) and receives a reward $R(t)$. For any horizon T, the goal for the learner is to attain the highest expected cumulative reward $[\sum_{t=1}^{T} R(t)]$. In this setup, the reward distribution of each arm $k \in [K]$ is assumed to be fixed over time and the rewards received by pulling the same arm are independently and identically distributed. An oracle (the best policy), in this case, would always play the arm with the highest mean reward. The difference between the expected cumulative reward obtained by the oracle and the one obtained by the learner is known to be the "regret." As is shown in existing literature [78], the regret lower bound for stochastic K-armed bandits is $\Omega(\sqrt{T})$. Many existing algorithms achieve the regret at the optimal rate $O(\sqrt{T})$, including the Upper Confidence Bound (UCB) algorithm and the Thompson Sampling (TS) algorithm. We also include a traditional algorithm Explore-then-Commit (ETC) and a greedy heuristic (ε-Greedy) in our study. These algorithms along with their regret guarantees are built upon the key assumption that the reward distributions are fixed. In Section 5.4, we test the validity of this assumption in a recommendation setting where the interactants are humans and the rewards represent their preferences.

Algorithms We give a brief summary and a high-level intuition for each algorithm. More detailed descriptions of these algorithms can be found in Appendix **??** in the supplementary material. We use the term "algorithm" in a broad sense and the following items (e.g., Self-selected and Fixed sequence) may not all be traditional MAB algorithms.

- Self-selected: The participants who are assigned to the Self-selected algorithm will choose which arm to interact with by themselves. In other words, at each time t, instead of a prescribed learning policy determining the arm $a(t)$, the participants themselves will choose the arm.
- UCB: At time t, UCB deterministically pulls the arm with the highest upper confidence bound, a value that for each arm, combines the empirical mean reward with an uncertainty estimate of the mean reward. An arm with high upper confidence bound can have high empirical mean and/or high uncertainty on the mean estimate.
- TS: In Thompson Sampling, a belief distribution is maintained over the possible reward values for each arm. At time t, the algorithm samples a reward from each arm's belief distribution and pulls the arm with the highest sampled reward. When a reward is received after pulling the arm, TS updates the corresponding arm's prior belief distribution to obtain a posterior.
- ETC: Unlike UCB and TS, the Explore-then-Commit algorithm has two separate stages—the exploration stage and the exploitation stage. It starts with an exploration period where the algorithm pulls the arms in a cyclic order and then switches to an exploitation stage where only the arm with the highest empirical mean in the exploration stage will be pulled. Given that the ETC algorithm achieves a regret of $O(T^{2/3})$ when the exploration time is on the order of $T^{2/3}$, we have set the exploration period to be $c \cdot T^{2/3}$ for a positive constant c.
- ε-Greedy: This greedy heuristic pulls the arm with the highest empirical mean with probability $1-\varepsilon$ where $0 < \varepsilon < 1$, and pulls an arm uniformly at random with probability ε. In a setting with long interaction period, one way of setting ε is to have it decreasing over time, e.g., setting ε to be on the order of $\frac{1}{t}$ [7]. Given the short interaction period in our setting, we have used a fixed $\varepsilon = 0.1$ (which may result in linear regret).
- Fixed sequence (CYCLE, REPEAT): The fixed sequence algorithms pull arms by following a predefined sequence. That is, the arm pulled at time t only depends on the current

time step and does not rely on the received rewards so far. We have used two fixed sequences CYCLE and REPEAT for testing whether the reward distributions (that represent participants' preferences) are fixed over time. We provide more details on these two fixed sequences in Section 5.4.

Next, we present how we have selected the comics used for recommendations.

5.3.2 Comics data

In our experiment, we choose comics as our recommendation domain for the following reasons: (i) Fast consumption time: Given the nature of our experiment where study participants are recommended a sequence of items to consume in a limited amount of time, we require the time for consuming each of the recommendations to be short. For example, recommending movie clips to watch may not be appropriate in our setting given that each clip may take a couple of minutes to finish. (ii) No strong pre-existing preferences: Another important feature of the chosen recommendation domain is that the majority of the study participants should have no strong preference on that subject prior to the experiment. For example, unlike comics, music is a subject that most people tend to already have strong preferences towards [126]. In such cases, the effects of recommendations towards the participants' preferences may be minimal.

We collected comics from 5 comic series on GoComics [47]. Each comic series belongs to a genre and represents an arm for pulling. The 5 comic series along with their genre are Lisa Benson (political, conservative), Nick Anderson (political, liberal), Baldo (family), The Born Loser (office), and The Argyle Sweater (gag). The genres of these comics are assigned by GoComics. For each series, we take the following steps to select the set of comics:

1. We first collect all comics belonging to the comic series from the year 2018. We select this time period to be not too recent so that the results of the study are not heavily influenced by ongoing events. It is also not too distant so that the content is still relevant to all subjects.
2. For each individual comic, we obtain its number of likes on GoComics. Then, we choose the top 60 comics from each comic genre/series in terms of the number of likes. This selection criteria is designed to ensure the quality of the chosen comics.
3. Finally, we randomly assign an ordering to the comics. We keep this ordering fixed throughout the study such that if an arm is pulled (a genre is chosen) at its j-th time, the presented comics will always be the same.

The comics within the same comic series are independent, in the sense that they can generally be read in any order, without requiring much context from previous comics from the same series. For these collected comics, we have labeled the number of unique characters in them, and use it for the attention check questions to ensure that the study participants have read the comics. Although we have adopted the above selection criteria to ensure the quality of comics from the same comic series to be similar, there is heterogeneity among individual comics and thus may influence the interpretation of our results. We provide more discussion on this in Section **??**. In Section 5.3.3, we discuss our experimental protocol and platform.

5.3.3 Experimental protocol and platform

We first outline our experimental protocol, which consists of the following steps (Figure 5.1):

1. *Background survey (initial filtering)*: We ask the study participants to complete a brief background survey. The participants will only be given a code to continue to the next step if an arithmetic question is answered correctly. The goal for the first step is to set up an initial filtering for participants.
2. *Registration*: After completing the background survey, participants are then asked to register on our platform using their completion code. Each participant in our study is assigned an algorithm in the following fixed probabilities—0.25 for Self-selected, 0.125 for UCB, 0.125 for TS, 0.125 for ETC, 0.125 for ε-Greedy and 0.125 for each of the two fixed sequences.
3. *Comic rating*: In this step, participants will read a sequence of 50 comics and provide an enjoyment score (a rating) after reading each comic. The sequence of comics can be generated by any of the algorithms discussed in Section 5.3.1. After reading each comic and providing a rating, the participants are also asked to answer an attention check question.
4. *Post-study survey*: Once the participants are finished with the comics rating step of the study, they are asked to complete a post-study survey about their reading experience. They are asked if they remember reading certain comics and if they have rated them positively, as well as how they perceive the recommendations they are provided. An example of the post-study survey questions can be found in Figure **??**.

Our experimental platform is built as a toolkit for running field tests for any MAB algorithms with human users. It consists of (a) the participant-facing web interface, and (b) the server backend that stores and processes incoming data from the web interface. When designing the experimental protocol and platform, we consider the following questions:

1. Given that we are asking users to give subjective feedback, how do we have more user responses that are reflective to the user's true preference?
2. How do we design an experimental interface that impose less bias to the users?
3. Since our study requires users to complete a long (up to 30-minute) sequence of non-independent tasks, how do we have the study to be less interrupted?
4. How do we build the system flexible enough to conduct studies for different MAB algorithms and test different assumptions of MAB setups, including ones that we do not cover in this work?

For (1), we adopt a 9-point Likert scale so that the numbers are sufficiently distinguishable to the participants [33] and check whether users are paying sufficient attention during the study. In particular, we test the participants on objective properties of the comics they have read, e.g. the number of unique characters (with a face and/or body) in them. We also set a minimum time threshold of 10 seconds before each user response can be submitted so that users spend adequate time on each comic.

For (2), to ensure that the participant's rating is not biased towards (e.g., anchored on) the Likert scale slider's initial value, we set the slider to be transparent before the participant clicks on the scale. In addition, in the Self-selected setting where the participants choose the genre of

	Self-selected	UCB	TS	ETC	ε-Greedy	CYCLE	REPEAT
# of Participants	74	40	44	39	41	40	38

Table 5.1: Number of participants for each algorithm. The description for Self-selected, UCB, TS, ETC and ε-Greedy are in Section 5.3.1. CYCLE and REPEAT are defined in Section 5.4.

comics to read next, we minimize the color- and ordering-based biases by setting the category selection buttons to be the same color and in random order.

As stated in design question (3), because we are interested in studying evolving preferences over a sequence of non-independent tasks, we would like to have *continuous and uninterrupted* user attention over a period of time. To do so, we design the system to be stateful so that participants can resume the study where they left off in the event of brief network disconnection and browser issues.

Finally, we discuss the flexibility of our system and address design question (4). Our experimental platform allows the experimenter to specify any recommendation domains and MAB algorithms for interacting with the human interactant. This flexibility allows the experimenter to not only test the performance of different MAB algorithms but also design pull sequences to understand user preference dynamics and test existing assumptions on user preferences. For example, one may design pull sequences to study the correlation between rewards obtained at different time steps and rewards obtained from pulling different but related arms. It is worth noting that the attention checks in our system are also customizable, allowing for diverse attention checks for each recommendation domain.

We have open sourced our code and data. For more details about the experimental platform and MTurk-related implementation details, see Section **??**.

5.3.4 Recruitment and compensation

To ensure the quality of our collected data, we only allow MTurk workers to participate in the study if they are U.S. residents and have completed at least 500 Human Intelligence Tasks (HITs) with an above 97% HIT approval rate. The anticipated (and actual) time to complete the study is less than 30 minutes. For participants who have successfully completed the study and answered the attention check questions correctly 70% of time, we have paid $\$7.5$ (the equivalent hourly salary is above $\$15/\text{hr}$). In Section **??**, we provide more details on the participants' demographics and backgrounds. Out of the 360 participants who have successfully completed the study, 316 passed the attention check (acceptance rate 87.8%). Our analyses are only conducted on the data collected from these participants. Table 5.1 shows the number of participants for each experimental setup.

5.4 Evolving Preferences in K-armed bandits

As we have previously discussed, in K-armed bandits, the reward distribution of each arm is assumed to be fixed over time [78, 121, 135], which implies that the mean reward of each arm remains the same. It is unclear whether such an assumption is reasonable in recommender system settings where the reward distributions represent human preferences. Our first aim is to test for the existence of evolving preference in the K-armed bandits setup. In other words, using randomized controlled trials, we want to test the following hypothesis: *In a K-armed bandit recommendation setting, the reward distribution of each arm (i.e., the user preference towards each item) is not fixed over time.*

To answer this, we collect enjoyment scores for two fixed recommendation sequences, where each sequence is of length $T = 50$. The sequence consists of recommendations from $K = 5$ genre of comics. In other words, the total number of arms is 5. The first sequence pulls the arms in a cyclic fashion which we denote by CYCLE, while the second sequence pulls each arm repeatedly for $m = T/K$ times which we denote by REPEAT:

$$\text{CYCLE}: \ (\underbrace{12\ldots K12\ldots K\ldots 12\ldots K}_{12\ldots K \text{ for } m \text{ times}}),$$
$$\text{REPEAT}: \ (\underbrace{22\ldots 2}_{m \text{ times}}\underbrace{1\ldots 1}_{m \text{ times}}3\ldots 3\ldots K\ldots K).$$

We note that for both sequences, the number of arm pulls of each arm is the same (m times). Since the order of the comics is fixed for each arm (e.g., pulling an arm for m times will always result in the same sequence of m comics from that genre), the set of comics recommended by the two sequences are the same. The only difference between the two sequences is the order of the presented comics. Intuitively, if the mean reward of each arm is fixed over time (and does not depend on the past pulls of that arm), then the (empirical) mean reward of each arm should be very similar under the two pull sequences.

In this work, we utilize a modification of the two-sample permutation test [38] to deal with the different numbers of participants under the two recommendation sequences. We let $CYCLE$ and $REPEAT$ denote the set of participants assigned to the CYCLE and REPEAT recommendation sequence, respectively. For each participant $i \in CYCLE$, we use $a_i(t)$ to denote the pulled arm (the recommended comic genre) at time t and $X_i(t)$ to denote the corresponding enjoyment score (the reward) that the participant has provided. Similarly, for each participant $j \in REPEAT$, we use $a_j(t)$ and $Y_j(t)$ to denote the arm pulled at time t and the enjoyment score collected from participant j at time t. Using these notations, for each arm $k \in [K]$, we define the test statistic as follows:

$$\tau_k = \underbrace{\frac{1}{|CYCLE|}\sum_{i \in CYCLE}\left(\frac{1}{m}\sum_{t \in [T]: a_i(t)=k} X_i(t)\right)}_{M_k^{\text{CYCLE}}} - \underbrace{\frac{1}{|REPEAT|}\sum_{j \in REPEAT}\left(\frac{1}{m}\sum_{t \in [T]: a_j(t)=k} Y_j(t)\right)}_{M_k^{\text{REPEAT}}}.$$

The test statistic τ_k is the difference between the mean reward (enjoyment score) M_k under the CYCLE recommendation sequence and the mean reward M_k under the REPEAT recommendation sequence for arm k. A non-zero τ_k suggests that the mean reward of arm k is different under

		family	gag	political (conservative)	office	political (liberal)
Overall	τ_k value	0.290	0.132	0.445	0.047	0.573
	95% CI	$[0.042, 0.549]$	$[-0.096, 0.371]$	$[0.158, 0.744]$	$[-0.196, 0.290]$	$[0.301, 0.837]$
	p-value	0.025^*	0.275	0.004^*	0.694	$< 0.001^*$
Heavy	τ_k value	-0.394	-0.556	-0.694	-0.647	-0.664
	95% CI	$[-0.678, -0.114]$	$[-0.864, -0.243]$	$[-1.056, -0.325]$	$[-0.944, -0.350]$	$[-1.021, -0.302]$
	p-value	0.007^*	$< 0.001^*$	$< 0.001^*$	$< 0.001^*$	$< 0.001^*$
Light	τ_k value	0.784	0.635	1.274	0.552	1.475
	95% CI	$[0.421, 1.150]$	$[0.298, 0.977]$	$[0.860, 1.681]$	$[0.198, 0.905]$	$[1.120, 1.827]$
	p-value	$< 0.001^*$	$< 0.001^*$	$< 0.001^*$	0.001^*	$< 0.001^*$

Table 5.2: The difference between the mean reward under and the mean reward under for each arm. All results are rounded to 3 digits. The p-values are obtained through permutation tests with $10,000$ permutations. We use asterisk to indicate that the test is significant at the level $\alpha = 0.1$. The 95% confidence intervals are obtained using bootstrap with $5,000$ bootstrapped samples.

and and that the reward distribution is evolving. The higher the absolute value of τ_k is, the bigger the difference between the two mean rewards is. A positive test statistic value indicates that the participants prefer the arm under over on average.

To quantify the significance of the value of the test statistic, we use a two-sample permutation test to obtain the p-value of the test [38]: First, we permute participants between and uniformly at random for $10,000$ times and ensure that the size of each group remains the same after each permutation. Then, we recompute the test statistic τ_k for each permutation to obtain a distribution of the test statistic. Finally, we use the original value of our test statistic along with this distribution to determine the p-value.

To report the 95% confidence interval of the test statistic for each arm, we use bootstrap and re-sample the data for $5,000$ times at the level of arm pulls. That is, for each arm pull at time t, we obtain its bootstrapped rewards under CYCLE and REPEAT by resampling from the actual rewards obtained by pulling that arm under CYCLE and REPEAT at time t, respectively. Given that we have conducted 5 tests simultaneously (one for each arm), in order to control the family-wise error rate, we need to correct the level α_k $(k \in [K])$ for each test. More formally, to ensure the probability that we falsely reject *any* null hypothesis to be at most α, for each test, the p-value of a test should be at most its corresponding corrected α_k. We adopt the Holm's Sequential Bonferroni Procedure (details presented in Section **??**) to perform this correction [2].

Our results show that for three arms—family, political (conservative) and political (liberal)—the non-zero difference between the mean reward under the two recommendation sequences are significant at level $\alpha = 0.1$ (Table 5.2). These findings confirm our research hypothesis that user preferences are not fixed (even in a short amount of time) in a K-armed bandit recommendation setting. There may be many causes of the evolving reward distributions (preferences). One possibility, among many others, is that the reward of an arm depends on its past pulls. In other words, people's preference towards an item depends on their past consumption of it. For example, existing marketing and psychology literature has suggested that people may experience hedonic decline upon repeated exposures to the same item [12, 41]. On the other hand, in music playlisting, one may expect the expected reward of an arm (a genre) to increase due to the taste that the listener has developed for that genre of music [126]. For a more comprehensive discussion on preference formation, we refer the readers to Becker [13].

Finally, to better understand the nature of our findings, we divide the participants into heavy comic readers who read comics daily and light comic readers who read comics at a lower frequency. Among participants who are assigned the CYCLE sequence, there are 17 heavy readers and 23 light readers. For REPEAT, there are 16 heavy readers and 22 light readers. We perform the same analysis as noted above for each of the two groups. Similar to the overall findings, among both heavy and light readers, evolving preferences (evolving mean reward) have been observed (Table 5.2), confirming our research hypothesis. Interestingly, we find that for each genre, the heavy readers tend to prefer the recommendations from the REPEAT sequence over the CYCLE sequence. On the contrary, for each genre, light readers prefer recommendations from the CYCLE sequence over the REPEAT sequence. As an initial step towards understanding this phenomenon, we present descriptive data analysis on how rewards (user preferences) evolve for heavy and light readers under the two recommendation sequences. Similar to our results in Table 5.2, for light readers, at most time steps, the reward trajectory of CYCLE has a higher value than the reward trajectory of REPEAT; while for heavy readers, this is the opposite (Figure 5.3). By looking at the reward trajectories (and the lines fitted through the reward trajectories) over the entire recommendation sequence (Figure 5.3), we find additional trends: for light readers, the differences between the rewards collected under CYCLE and REPEAT are increasing over time; while for heavy readers, such differences are relatively stable. This trend is also observed in the reward trajectory (and the line fitted through the reward trajectory) of each arm under CYCLE and REPEAT (Figure 5.4). In particular, for light readers, among all arms (comic genres) except family, we find that the lines fitted through the rewards collected under CYCLE and REPEAT become further apart as the number of arm pulls increases. This distinction between heavy and light users may be due to various reasons. For example, light readers may prefer variety in their recommendations because they are exploring their interests or the light readers and heavy readers share different satiation rates. On a related note, a recent study has shown that the satiation rates of people may depend on their personality traits [42]. Precisely understanding the causes of this distinction between heavy and light readers is of future interest.

5.5 Usage Example of the Experimental Framework

As an illustration of the usage of our experimental framework, we compare the performance of different algorithms, in terms of both the cumulative rewards, and the users' own reflection on their interactions with these algorithms. Though we are in a simulated and simplified recommender system setup, we aim to provide some understanding on (i) whether people prefer to be recommended by an algorithm over deciding on their own, and (ii) whether more autonomy (choosing the next comic genre on their own) results in a more attentive and mindful experience. We note that, compared to our findings in Section 5.4 that are obtained through a rigorous hypothesis testing framework, the results in this section are exploratory in nature and should not be interpreted as definitive answers to the above questions.

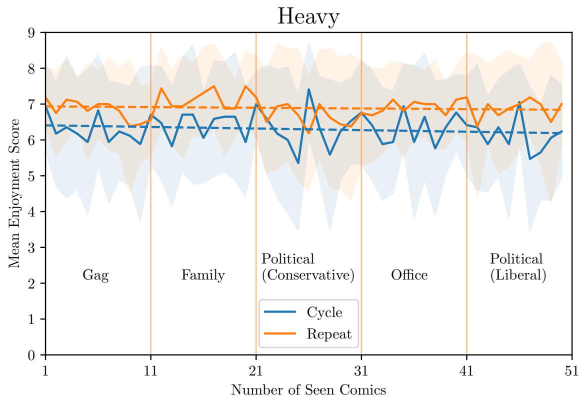


Figure 5.3: The reward at each time step of the CYCLE and REPEAT recommendation sequence averaged across heavy and light readers, respectively. The error bars indicate one standard deviation from the mean. For the REPEAT sequence, we highlight when the arm switches using the vertical lines and add the arm name (comic genre) corresponding to the time period in between the switches using the black texts. The blue and orange dotted lines are fitted through the rewards collected under CYCLE and REPEAT, respectively.

5.5.1 Enjoyment

We first compare different algorithms (Self-selected, UCB, TS, ETC, ε-Greedy, CYCLE and REPEAT) in terms of the participants' enjoyment. More specifically, we want to compare the participants who are provided recommended comics to read with the ones who choose on their own. In general, enjoyment is hard to measure [112]. To this end, we look at participants' enjoyment and preference towards these algorithms through the following three aspects:

- Cumulative rewards: This metric is closely related to the notion of "regret" that is commonly used to compare the performance of bandit algorithms [78]. More formally, for any participant i, the cumulative reward of an algorithm that interacts with participant i is given by $\sum_{t=1}^{T} R_i(t)$ where $R_i(t)$ is the reward provided by participant i at time t. In Table 5.3,

	Self-selected	UCB	TS	ETC	ε-Greedy	CYCLE	REPEAT
Cumulative reward	319.36	323.70	312.37	324.49	307.59	316.5	301.63
	[205, 434]	[223, 424]	[204, 420]	[209, 440]	[205, 411]	[216, 417]	[201, 427]
Hindsight satisfaction	81.08%	75.00%	72.73%	76.92%	80.49%	77.50%	63.16%
Preference towards autonomy	71.62%	50.00%	68.18%	58.97%	58.54%	62.50%	65.79%

Table 5.3: Performances of each algorithm in terms of different enjoyment characterizations. The first row gives the cumulative rewards for each algorithm, averaged over participants who have interacted with it. We also report the 95% confidence intervals for the cumulative rewards. The hindsight-satisfaction row shows the percentage of participants who believe the sequence of comics they have read captures their preference well. The last row provides the percentage of participants who prefer to select comics to read on their own in hindsight.

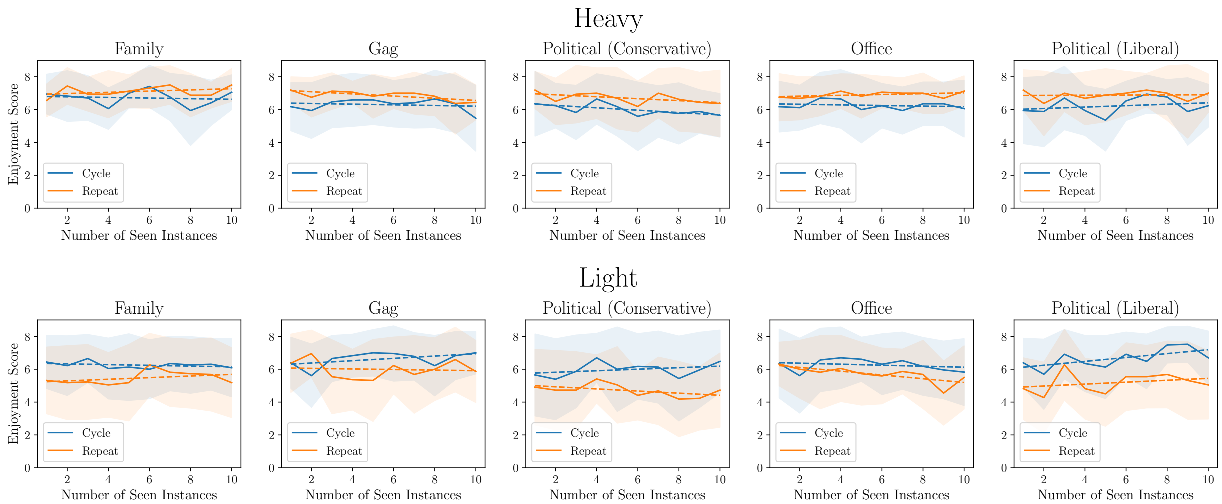


Figure 5.4: Each plot shows the reward collected for a particular arm at each arm pull under the CYCLE and REPEAT recommendation sequences. The error bars indicate one standard deviation from the mean. The reward trajectories are averaged across heavy and light readers, respectively. The blue and orange dotted lines are fitted through the reward trajectories for each arm under CYCLE and REPEAT, respectively.

for each algorithm, we show their cumulative rewards averaged over participants who have interacted with the algorithm.

- Hindsight satisfaction: After the participants interact with the algorithm, in the post-study survey, we ask them the following question: "Do you feel that the sequence of recommendations[2] captured your preferences well?" Participants' answers to this question provide their hindsight reflections towards how well the algorithm performed and whether they are satisfied with it. The second row of Table 5.3 shows the percentage of participants who believe the sequence of comics they read has captured their preference well.
- Preference towards autonomy: In addition to the previous two metrics, we have explicitly asked the participants to indicate whether they prefer to choose comics to read on their own. In the post-study survey, we ask: "Do you prefer being recommended comics to read or selecting comics to read on your own?" The third row of Table 5.3 provides the percentage of participants who prefer to select comics on their own for each algorithm.

In our collected data, the participants who have given more autonomy (the Self-selected participants) prefer more autonomy in hindsight, compared to other participants. Though Self-selected does not have the highest mean cumulative reward, it has the highest percent of participants who believe that the comics they read have captured their preferences well in hindsight (Table 5.3). This misalignment between mean cumulative reward and hindsight satisfaction also shows in other algorithms (i.e., higher mean cumulative reward may not indicate higher hindsight satisfaction), including ε-Greedy and ETC. On the other hand, UCB performs well in terms of the mean cumulative reward, while having the lowest percentage of participants wanting to

[2] For Self-selected participants, this should be interpreted as comics they have read/self-selected.

	Self-selected	UCB	TS	ETC	ε-Greedy	CYCLE	REPEAT
Reading Memory	91.89%	90.00%	92.42%	93.16%	90.24%	90.00%	90.35%
Rating Memory	71.17%	70.00%	70.45%	76.92%	69.92%	69.16%	57.89%

Table 5.4: Average correctness of the two types of memory questions for each algorithm.

have more autonomy. We provide additional analyses on comparing these algorithms in terms of participant's enjoyment in Section **??**.

5.5.2 Attentiveness

We want to understand whether participants who are asked to choose on their own have a more mindful experience, in which the participants are more attentive to what they have read through [36]. There is no consensus on defining and measuring mindfulness of an experience [50]. Some of the existing research uses self-reported mindfulness as a measurement [49]. In our case, we look at how attentive the participants are to their own experience, though the lens of memory—for each participant, in the post-study survey, we ask them two types of memory questions:

- Reading memory: In the post-study survey, we present the participants three randomly selected comics and ask them to indicate whether they have read the comics before. This question aims to measure the participants' memory of *what they have read*. The first row of Table 5.4 shows the average correctness percentage for the reading memory questions for each algorithm.
- Rating memory: We also look at the participants' memory on *how they have liked* certain comics. To this end, in the post-study survey, we present three randomly selected comics among the ones the participants have read and ask them to indicate whether they have rated the comic with a score of five or above. The second row of Table 5.4 shows the average correctness percentage.

We note that these questions differ from the attention check questions after each comic and are not directly related to the compensation that the participants get. However, the high correctness percentage shown in Table 5.4 on the reading memory questions suggest that the participants have attempted to answer the questions from their memory instead of providing random answers.

In general, the participants have performed much better on the reading memory questions than the rating memory questions, suggesting that they may be more aware of what they have consumed than how they have liked them. In addition, all participants perform similarly in terms of the reading memory correctness percentage with those whose algorithm is ETC or Self-selected performing slightly better. Though it is hard to say which algorithm provides the most attentive experience to the participants and whether Self-selected participants have a more mindful experience, it is relatively clear that participants whose algorithm is REPEAT perform the worst in terms of rating memory. Our data does not provide strong evidence on believing that more autonomy results in more attentive experience, but may suggest that less enjoyable experience (e.g., for participants whose algorithm are REPEAT) correlates to less attentiveness. We provide additional analyses on comparing these algorithms in terms of participant's attentiveness in Appendix **??**.

Chapter 6

Conclusions and Future Directions

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association. URL `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`. 3.5

[2] Hervé Abdi. Holm's sequential bonferroni procedure. *Encyclopedia of research design*, 1 (8):1–8, 2010. 5.4

[3] Daniel Acuna and Paul Schrater. Bayesian modeling of human sequential decision-making on the multi-armed bandit problem. In *Proceedings of the 30th annual conference of the cognitive science society*, volume 100, pages 200–300. Washington, DC: Cognitive Science Society, 2008. 5.2

[4] Leonard Adolphs, Michelle Chen Huebscher, Christian Buck, Sertan Girgin, Olivier Bachem, Massimiliano Ciaramita, and Thomas Hofmann. Decoding a neural retriever's latent space for query suggestion. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8786–8804, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.601. URL `https://aclanthology.org/2022.emnlp-main.601`. 4.2

[5] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. *Mining text data*, pages 77–128, 2012. 4.1

[6] Christoph Albrecht, Arif Merchant, Murray Stokely, Muhammad Waliji, Francois Labelle, Nathan Coehlo, Xudong Shi, and Eric Schrock. Janus: Optimal flash provisioning for cloud storage workloads. In *Proceedings of the USENIX Annual Technical Conference*, pages 91–102, 2560 Ninth Street, Suite 215, Berkeley, CA 94710, USA, 2013. URL `https://www.usenix.org/system/files/conference/atc13/atc13-albrecht.pdf`. 3.6.4, 3.6.4

[7] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002. 5.3.1

[8] Harriet E Baber. Adaptive preference. *Social Theory and Practice*, 33(1):105–126, 2007.

5.2

[9] Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan. Magpie: Online modelling and performance-aware systems. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9*, 2003. 3.1, 3.2

[10] Andrea Barraza-Urbina and Dorota Glowacka. Introduction to bandits in recommender systems. In *Fourteenth ACM Conference on Recommender Systems*, pages 748–750, 2020. 5.1

[11] Soumya Basu, Rajat Sen, Sujay Sanghavi, and Sanjay Shakkottai. Blocking bandits. In *Advances in Neural Information Processing Systems*, pages 4785–4794, 2019. 5.2

[12] Manel Baucells and Rakesh K Sarin. Satiation in discounted utility. *Operations research*, 55(1):170–181, 2007. 5.2, 5.4

[13] Gary S Becker. *Accounting for tastes*. Harvard University Press, 1996. 5.4

[14] N. Beckmann and D. Sanchez. Maximizing cache performance under uncertainty. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 109–120, 2017. 3.1, 3.3.2, 3.3.4, 3.6.4

[15] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007. 5.1

[16] Christopher M. Bishop. Mixture density networks. 1994. URL `http://publications.aston.ac.uk/id/eprint/373/`. 2, 3.4.3

[17] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in neural information processing systems*, 29, 2016. 4.5.2

[18] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022. 4.1

[19] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1002. URL `https://aclanthology.org/K16-1002`. 4.2

[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 4.2

[21] Leonardo Cella and Nicolò Cesa-Bianchi. Stochastic bandits with delay-dependent payoffs. In *International Conference on Artificial Intelligence and Statistics*, pages 1168–1177, 2020. 5.2

[22] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert Henry, Robert Bradshaw, and Nathan. Flumejava: Easy, efficient data-parallel pipelines. In *ACM*

SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 363–375, 2 Penn Plaza, Suite 701 New York, NY 10121-0701, 2010. URL `http://dl.acm.org/citation.cfm?id=1806638`. 3.5

[23] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26 (2):4, 2008. 3.2, 3.3.2

[24] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 456–464, 2019. 5.1, 5.2

[25] Minmin Chen, Bo Chang, Can Xu, and Ed H Chi. User response models to improve a reinforce recommender system. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 121–129, 2021. 5.1, 5.2

[26] Minmin Chen, Yuyan Wang, Can Xu, Ya Le, Mohit Sharma, Lee Richardson, Su-Lin Wu, and Ed Chi. Values of user exploration in recommender systems. In *Fifteenth ACM Conference on Recommender Systems*, pages 85–95, 2021. 5.1, 5.2

[27] Minmin Chen, Can Xu, Vince Gatto, Devanshu Jain, Aviral Kumar, and Ed Chi. Off-policy actor-critic for recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 338–349, 2022. 5.1, 5.2

[28] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. 1

[29] Larry Christensen and Alisa Brooks. Changing food preference as a function of mood. *The journal of psychology*, 140(4):293–306, 2006. 5.2

[30] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):8, 2013. 3.1

[31] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 153–167, New York, NY, USA, 2017. Association for Computing Machinery. URL `https://doi.org/10.1145/3132747.3132772`. 3.1, 3.2, 3.4.2

[32] William Coster and David Kauchak. Simple English Wikipedia: A new text simplification task. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 665–669, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `https://aclanthology.org/P11-2117`. 4.4.2

[33] Eli P Cox III. The optimal number of response alternatives for a scale: A review. *Journal of marketing research*, 17(4):407–422, 1980. 5.1, 5.3.3

[34] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 127–144, New York, NY, USA, 2014. Association for Computing Machinery. URL `https://doi.org/10.1145/2541940.2541941`. 3.2

[35] 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. 3.2

[36] Ronald M Epstein. Mindful practice. *Jama*, 282(9):833–839, 1999. 5.5.2

[37] Tamara Fernandez, Nicolas Rivera, and Yee Whye Teh. Gaussian processes for survival analysis. In *Advances in Neural Information Processing Systems 29*, pages 5021–5029. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6443-gaussian-processes-for-survival-analysis.pdf`. 3.3.4

[38] Ronald Aylmer Fisher. Design of experiments. *Br Med J*, 1(3923):554–554, 1936. 5.4

[39] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004. 3.1

[40] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–28, 1994. 3.7

[41] Jeff Galak and Joseph P Redden. The properties and antecedents of hedonic decline. *Annual review of psychology*, 69:1–25, 2018. 5.1, 5.2, 5.4

[42] Jeff Galak, Jinwoo Kim, and Joseph P Redden. Identifying the temporal profiles of hedonic decline. *Organizational Behavior and Human Decision Processes*, 169:104128, 2022. 5.4

[43] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, Kelvin Hu, Meghna Pancholi, Yuan He, Brett Clancy, Chris Colen, Fukang Wen, Catherine Leung, Siyuan Wang, Leon Zaruvinsky, Mateo Espinosa, Rick Lin, Zhongling Liu, Jake Padilla, and Christina Delimitrou. An open-source benchmark suite for microservices and their hardware-software implications for cloud edge systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 3–18, New York, NY, USA, 2019. Association for Computing Machinery. URL `https://doi.org/10.1145/3297858.3304013`. 3.1, 3.3.1

[44] Yu Gan, Yanqi Zhang, Kelvin Hu, Dailun Cheng, Yuan He, Meghna Pancholi, and Christina Delimitrou. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 19–33, New York, NY, USA, 2019. Association for Computing Machinery. URL `https://doi.org/10.1145/3297858.3304004`. 3.2

[45] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003. 1.1, 3.1

[46] Daniel Gillick, Alessandro Presta, and Gaurav Singh Tomar. End-to-end retrieval in continuous space. *arXiv preprint arXiv:1811.08008*, 2018. 4.1

[47] GoComics. Gocomics. `https://www.gocomics.com`, 2021. 5.3.2

[48] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015. 5.1

[49] Laurie A Greco, Ruth A Baer, and Gregory T Smith. Assessing mindfulness in children and adolescents: development and validation of the child and adolescent mindfulness measure (camm). *Psychological assessment*, 23(3):606, 2011. 5.5.2

[50] Paul Grossman. On measuring mindfulness in psychosomatic and psychological research. 2008. 5.5.2

[51] Philip M Groves and Richard F Thompson. Habituation: a dual-process theory. *Psychological review*, 77(5):419, 1970. 5.1

[52] Faruk Gul and Wolfgang Pesendorfer. The revealed preference theory of changing tastes. *The Review of Economic Studies*, 72(2):429–448, 2005. 5.2

[53] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020. URL `https://arxiv.org/abs/1908.10396`. 3.5

[54] Evan Hernandez, Belinda Z Li, and Jacob Andreas. Inspecting and editing knowledge representations in language models. *arXiv preprint arXiv:2304.00740*, 2023. 4.2

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3.2

[56] Chester Holtz, Chao Tao, and Guangyu Xi. BanditPyLib: a lightweight python library for bandit algorithms. Online at: `https://github.com/Alanthink/banditpylib`, 2020. URL `https://github.com/Alanthink/banditpylib`. Documentation at `https://alanthink.github.io/banditpylib-doc`. 5.2

[57] John H Howard, Michael L Kazar, Sherri G Menees, David A Nichols, Mahadev Satyanarayanan, Robert N Sidebotham, and Michael J West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988. 3.1

[58] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2553–2561, 2020. 4.1, 4.2

[59] Md Hussain and Ishtiak Mahmud. pymannkendall: a python package for non parametric mann kendall family of trend tests. *Journal of Open Source Software*, 4(39):1556, 2019.

5

[60] Sarika Jain, Anjali Grover, Praveen Singh Thakur, and Sourabh Kumar Choudhary. Trends, problems and solutions of recommender system. In *International conference on computing, communication & automation*, pages 955–958. IEEE, 2015. 5.1

[61] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. High performance cache replacement using re-reference interval prediction (rrip). *SIGARCH Comput. Archit. News*, 38(3):60–71, June 2010. URL `https://doi.org/10.1145/1816038.1815971`. 3.1, 3.3.2, 3.6.4

[62] Yibo Jiang, Goutham Rajendran, Pradeep Ravikumar, Bryon Aragam, and Victor Veitch. On the origins of linear representations in large language models. *arXiv preprint arXiv:2403.03867*, 2024. 4.1

[63] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 781–789, 2017. 5.1

[64] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017. URL `https://doi.org/10.1145/3140659.3080246`. 3.5

[65] Svilen Kanev, Juan Pablo Darago, Kim Hazelwood, Parthasarathy Ranganathan, Tipp Moseley, Gu-Yeon Wei, and David Brooks. Profiling a warehouse-scale computer. *ACM SIGARCH Computer Architecture News*, 43(3):158–169, 2016. 3.1

[66] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[67] Komal Kapoor, Karthik Subbian, Jaideep Srivastava, and Paul Schrater. Just in time recommendations: Modeling the dynamics of boredom in activity streams. In *Proceedings of the eighth ACM international conference on web search and data mining*, pages 233–242, 2015. 5.2

[68] Rafael-Michael Karampatsis and Charles Sutton. Maybe deep neural networks are the

best choice for modeling source code. *arXiv preprint arXiv:1903.05734*, 2019. 3.3.3

[69] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL `https://aclanthology.org/2020.emnlp-main.550`. 4.2

[70] Divyansh Kaushik, Eduard Hovy, and Zachary Lipton. Learning the difference that makes a difference with counterfactually-augmented data. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Sklgs0NFvr`. 4.4.1

[71] Maurice George Kendall. Rank correlation methods. 1948. 4.5.2

[72] Taejin Kim, Sangwook Shane Hahn, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. Pcstream: Automatic stream allocation using program contexts. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, Boston, MA, July 2018. USENIX Association. URL `https://www.usenix.org/conference/hotstorage18/presentation/kim-taejin`. 3.1, 3.3.2

[73] Vadim Kirilin, Aditya Sundarrajan, Sergey Gorinsky, and Ramesh K. Sitaraman. Rl-cache: Learning-based cache admission for content delivery. In *Proceedings of the 2019 Workshop on Network Meets AI ML*, NetAI'19, page 57–63, New York, NY, USA, 2019. Association for Computing Machinery. URL `https://doi.org/10.1145/3341216.3342214`. 3.2

[74] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. *Advances in neural information processing systems*, 28, 2015. 4.2

[75] Robert Kleinberg and Nicole Immorlica. Recharging bandits. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 309–319. IEEE, 2018. 5.2

[76] Kai Konen, Sophie Jentzsch, Diaoulé Diallo, Peer Schütt, Oliver Bensch, Roxanne El Baff, Dominik Opitz, and Tobias Hecking. Style vectors for steering generative large language models. In Yvette Graham and Matthew Purver, editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 782–802, St. Julian's, Malta, March 2024. Association for Computational Linguistics. URL `https://aclanthology.org/2024.findings-eacl.52`. (document), 4.2, 4.5.1, **??**, 4.9

[77] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 5.1

[78] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020. 5.1, 5.3.1, 5.4, 5.5.1

[79] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014. 4.2

[80] Michael D Lee, Shunan Zhang, Miles Munro, and Mark Steyvers. Psychological models of human and optimal performance in bandit problems. *Cognitive Systems Research*, 12 (2):164–174, 2011. 5.2

[81] Pei Lee, Laks VS Lakshmanan, Mitul Tiwari, and Sam Shah. Modeling impression discounting in large-scale recommender systems. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1837–1846, 2014. 5.2

[82] Damien Lefortier, Adith Swaminathan, Xiaotao Gu, Thorsten Joachims, and Maarten de Rijke. Large-scale validation of counterfactual learning methods: A test-bed. *arXiv preprint arXiv:1612.00367*, 2016. 5.2

[83] Eric Lehman, Sarthak Jain, Karl Pichotta, Yoav Goldberg, and Byron Wallace. Does BERT pretrained on clinical notes reveal sensitive data? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 946–959, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.73. URL `https://aclanthology.org/2021.naacl-main.73`. 4.1

[84] Liu Leqi, Fatma Kilinc-Karzan, Zachary C Lipton, and Alan L Montgomery. Rebounding bandits for modeling satiation effects. 2021. 5.2

[85] Nir Levine, Koby Crammer, and Shie Mannor. Rotting bandits. In *Advances in neural information processing systems*, pages 3074–3083, 2017. 5.2

[86] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. 4.1

[87] Chunyuan Li, Xiang Gao, Yuan Li, Baolin Peng, Xiujun Li, Yizhe Zhang, and Jianfeng Gao. Optimus: Organizing sentences via pre-trained modeling of a latent space. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4678–4699, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.378. URL `https://aclanthology.org/2020.emnlp-main.378`. 4.2

[88] Haoran Li, Mingshi Xu, and Yangqiu Song. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 14022–14040, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.881. URL `https://aclanthology.org/2023.findings-acl.881`. 4.1, 4.2

[89] Jiwei Li, Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1106–1115, Beijing, China,

July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1107. URL `https://aclanthology.org/P15-1107`. 4.2

[90] 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. 5.1

[91] Evan Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. An imitation learning approach for cache replacement. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6237–6247. PMLR, 13–18 Jul 2020. URL `http://proceedings.mlr.press/v119/liu20f.html`. 3.2

[92] Fenrong Liu et al. *Changing for the better: Preference dynamics and agent diversity*. Institute for Logic, Language and Computation, 2007. 5.2

[93] Sheng Liu, Lei Xing, and James Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*, 2023. 4.2

[94] Kaiji Lu, Piotr Mardziel, Fangjing Wu, Preetam Amancharla, and Anupam Datta. Gender bias in neural natural language processing. In *Logic, Language, and Security*, pages 189–202. Springer, 2020. 4.5.2

[95] Henry B Mann. Nonparametric tests against trend. *Econometrica: Journal of the econometric society*, pages 245–259, 1945. 4.5.2

[96] Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. *arXiv preprint arXiv:2310.06824*, 2023. 4.1

[97] Julian John McAuley and Jure Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908. ACM, 2013. 5.1

[98] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson, and Rishabh Mehrotra. Explore, exploit, and explain: personalizing explainable recommendations with bandits. In *Proceedings of the 12th ACM conference on recommender systems*, pages 31–39, 2018. 5.1, 5.2

[99] Rishabh Mehrotra, Niannan Xue, and Mounia Lalmas. Bandit based optimization of multiple objectives on a music streaming platform. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3224–3233, 2020. 5.1, 5.2

[100] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 3.4.3, 4.1, 4.2, 4.3, 4.5.2

[101] Yonatan Mintz, Anil Aswani, Philip Kaminsky, Elena Flowers, and Yoshimi Fukuoka. Nonstationary bandits with habituation and recovery dynamics. *Operations Research*, 68 (5):1493–1516, 2020. 5.2

[102] Ivan Montero, Nikolaos Pappas, and Noah A. Smith. Sentence bottleneck autoencoders from transformer language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1822–1831, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.137. URL `https://aclanthology.org/2021.emnlp-main.137`. (document), 4.2, 4.4.1, 4.5.1, **??**, 4.9

[103] John Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander Rush. Text embeddings reveal (almost) as much as text. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12448–12460, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.765. URL `https://aclanthology.org/2023.emnlp-main.765`. 1.2, 4.1, 4.2, 4.3, 4.3.1, 1, 4.3.3

[104] R. E. Mullen. The lognormal distribution of software failure rates: application to software reliability growth modeling. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pages 134–142, 1998. 3.3.4

[105] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022. 4.1, 4.2

[106] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human-generated MAchine reading COmprehension dataset, 2017. URL `https://openreview.net/forum?id=Hk1iOLcle`. 4.3.3

[107] Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, and Yinfei Yang. Large dual encoders are generalizable retrievers. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9844–9855, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.669. URL `https://aclanthology.org/2022.emnlp-main.669`. 4.1, 4.2

[108] OpenCensus. Opencensus. `https://opencensus.io/`, 2020. URL `https://opencensus.io/`. 3.1, 3.3.1

[109] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The ramcloud storage system. *ACM Trans. Comput. Syst.*, 33(3), August 2015. URL `https://doi.org/10.1145/2806887`. 3.3.2

[110] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 3sigma: Distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. URL `https://doi.org/10.1145/3190508.3190515`. 3.1, 3.2, 3.2, 3.3.4, 3.4.2

[111] Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothe-

sis and the geometry of large language models. In *Causal Representation Learning Workshop at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=T0PoOJg8cK. 4.1

[112] John W Payne, James R Bettman, David A Schkade, Norbert Schwarz, and Robin Gregory. Measuring constructed preferences: Towards a building code. In *Elicitation of preferences*, pages 243–275. Springer, 1999. 5.5.1

[113] Drazen Prelec. Decreasing impatience: a criterion for non-stationary time preference and "hyperbolic" discounting. *Scandinavian Journal of Economics*, 106(3):511–532, 2004. 5.2

[114] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019. 3.2

[115] Dimitrios Rafailidis and Alexandros Nanopoulos. Modeling users preference dynamics and side information in recommender systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(6):782–792, 2015. 5.2

[116] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020. 1

[117] Ori Ram, Liat Bezalel, Adi Zicher, Yonatan Belinkov, Jonathan Berant, and Amir Globerson. What are you token about? dense retrieval as distributions over the vocabulary. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2481–2498, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.140. URL https://aclanthology.org/2023.acl-long.140. 4.2

[118] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410. 4.2

[119] Paul B Reverdy, Vaibhav Srivastava, and Naomi Ehrich Leonard. Modeling human decision making in generalized gaussian multiarmed bandits. *Proceedings of the IEEE*, 102(4):544–571, 2014. 5.2

[120] Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*, 2023. 4.2

[121] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952. 5.1, 5.4

[122] Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. Gen-

der bias in coreference resolution. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 8–14, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2002. URL `https://aclanthology.org/N18-2002`. 4.5.2

[123] S. R. Safavian and D. Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674, 1991. 3.4.1

[124] Yuta Saito, Aihara Shunsuke, Matsutani Megumi, and Narita Yusuke. Open bandit dataset and pipeline: Towards realistic and reproducible off-policy evaluation. *arXiv preprint arXiv:2008.07146*, 2020. 5.2

[125] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022. 1

[126] Thomas Schäfer and Peter Sedlmeier. What makes us like music? determinants of music preference. *Psychology of Aesthetics, Creativity, and the Arts*, 4(4):223, 2010. 5.3.2, 5.4

[127] Denis Serenyi. PDSW-DISCS Keynote: From GFS to Colossus: Cluster-Level Storage at Google. `http://www.pdsw.org/pdsw-discs17/slides/PDSW-DISCS-Google-Keynote.pdf`. 2017. 3.2, 3.3.3

[128] Julien Seznec, Andrea Locatelli, Alexandra Carpentier, Alessandro Lazaric, and Michal Valko. Rotting bandits are no harder than stochastic ones. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2564–2572, 2019. 5.2

[129] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005. 5.1, 5.2

[130] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. *Advances in neural information processing systems*, 30, 2017. 2, 4.4.1

[131] Tianxiao Shen, Jonas Mueller, Regina Barzilay, and Tommi Jaakkola. Educating text autoencoders: Latent representation guidance via denoising. In *International conference on machine learning*, pages 8719–8729. PMLR, 2020. (document), 4.2, 4.4.1, 4.5.1, **??**, 4.9

[132] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. On-the-fly adaptation of source code models using meta-learning. *arXiv preprint arXiv:2003.11768*, 2020. 3.3.3

[133] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop distributed file system. In *MSST*, volume 10, pages 1–10, 2010. 3.1, 3.2

[134] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. 2010. URL `https://research.google/pubs/pub36356/`. 3.1, 3.2

[135] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends®*

in Machine Learning, 12(1-2):1–286, 2019. 5.1, 5.4

[136] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 377–390, 2020. 4.1

[137] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 529–544, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/song. 3.2

[138] Michael Stonebraker and Greg Kemnitz. The postgres next generation database management system. *Communications of the ACM*, 34(10):78–92, 1991. 3.1

[139] Nishant Subramani, Nivedita Suresh, and Matthew Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 566–581, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.48. URL https://aclanthology.org/2022.findings-acl.48. (document), 4.2, 4.4.1, 4.5.1, **??**, 4.9

[140] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015. 5.1

[141] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 4.5.1

[142] Curt Tigges, Oskar John Hollinsworth, Atticus Geiger, and Neel Nanda. Linear representations of sentiment in large language models. *arXiv preprint arXiv:2310.15154*, 2023. 4.1

[143] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 4.3.3

[144] William T Tucker. The development of brand loyalty. *Journal of Marketing research*, 1 (3):32–35, 1964. 5.2

[145] Alex Turner, Lisa Thiergart, David Udell, Gavin Leech, Ulisse Mini, and Monte MacDiarmid. Activation addition: Steering language models without optimization. *arXiv preprint arXiv:2308.10248*, 2023. 4.2

[146] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/

2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. 4.2

[147] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 3.2, 1

[148] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018. 3.5

[149] Kexin Wang, Nils Reimers, and Iryna Gurevych. Tsdae: Using transformer-based sequential denoising auto-encoder for unsupervised sentence embedding learning. *arXiv preprint arXiv:2104.06979*, 2021. 4.1

[150] Yuyan Wang, Mohit Sharma, Can Xu, Sriraj Badam, Qian Sun, Lee Richardson, Lisa Chung, Ed H Chi, and Minmin Chen. Surrogate for long-term user experience in recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4100–4109, 2022. 5.1, 5.2

[151] Zhao Wang and Aron Culotta. Robustness to spurious correlations in text classification via automatically generated counterfactuals. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14024–14031, 2021. 4.4.1

[152] Jian Wei, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications*, 69:29–39, 2017. 5.1

[153] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, page 307–320, USA, 2006. USENIX Association. 3.1, 3.2

[154] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 3.7

[155] Wei Xu, Alan Ritter, Bill Dolan, Ralph Grishman, and Colin Cherry. Paraphrasing for style. In *Proceedings of COLING 2012*, pages 2899–2914, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL https://aclanthology.org/C12-1177. 4.4.2

[156] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound ai systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/, 2024. 1

[157] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022. 1

[158] Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to ai transparency. *arXiv preprint arXiv:2310.01405*, 2023. 4.2