Mitigating fragility in machine learning systems using structured models

Giulio Zhou

September 2023

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 applications in a variety of complex settings, such as web-scale search, content recommendation, autonomous vehicles, and language-based digital assistants. These *ML systems* have, in recent years, become predominantly data-driven, often underpinned by deep learning models that can effectively learn complex functions end-to-end from large amounts of available data. Since they make few assumptions on the learned function, neural networks are a flexible and effective tool for a wide range of tasks and environments. However, their purely data-driven nature also makes the learned solutions opaque, sample inefficient, and potentially brittle. To address these problems, there has been much work on imposing structure into ML models. Some approaches impose structure implicitly, such as through architecture design and data augmentation. Other approaches instead opt to impose structure explicitly, such as by incorporating latent priors, geometric constraints, and physical models. When suitably applied, imposing explicit structure is beneficial since it allows one to take advantage of the powerful learning capabilities of deep learning models while avoiding the shortcomings of their end-to-end and blackbox nature. Imposing such structure into models also improves their transparency and yields a framework to represent various characteristics integral to the modeled task or environment, such as variability, periodicity, stationarity, smoothness, and monotonicity.

In this thesis, we explore approaches to improving the reliability and transparency of ML systems by imposing explicit structure in the form of simple parametric models. Compared to previous approaches that incorporate known domain-specific structure (e.g. based on physical models), we show that simple parametric models are widely applicable, i.e. to any system whose behavior can be well-approximated by these models. We explore this using three case studies in different ML systems. In our first work, we model the variability of large warehouse-scale storage systems using univariate log-normal distributions (parametrized by neural networks), and show that doing so enables building an effective ML-driven storage system. In our second work, we validate the stationarity (of the reward distributions) within bandit-based recommender systems. Using human studies conducted on Mechanical Turk, we find evidence that stationarity assumptions do not hold, demonstrating the importance of validating the structures underlying ML systems. Lastly, for our proposed work, we analyze the benefits of imposing latent space structure in variational auto-encoders (VAEs) models of text. To improve the reliability of semantic transformations such as tense and sentiment, we propose quantifying the isotropy and smoothness of the VAE latent space and exploring transformation methods that take advantage of its unique geometry.

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 smart 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, state-of-the-art language models such as GPT-3 [22], PaLM [31], OPT-175B [160], and BLOOM [128] containing hundreds of billions of parameters have been trained using exaflop-scale compute clusters on the timescale of weeks and months. The growing capabilities of large models has shown that deep learning models can reliably learn general functions given sufficiently large amounts of data [75]. For many settings, however, it is not feasible and sample efficient to obtain or train on this quantity of data, and the resulting models are often brittle, as well as difficult to interpret.

To improve the efficiency of the learning process and the reliability of the learned representations, there has been a substantial amount of work on incorporating structure into ML models. Through a statistical machine learning lens, adding structure is analogous to adding bias into the learner in order to decrease the variance (and analogously the brittleness) of the learned representation. While this analysis does not strictly hold for deep learning models [14], properly constraining the model space (through imposing structure) intuitively yields more reliable solutions that are easier to learn. In practice, imposing structure in ML models can done either implicitly or explicitly. Implicit structure is commonly applied in the form of architecture design, regularization, or data augmentation that add functional biases and invariances, without modifying the typical end-to-end nature of deep learning models. On the other hand, approaches that impose explicit structure have been highly successful in problems where the system being modeled has known structure. These include incorporating explicit structure in the form of models of the physical world [36, 56, 162] as well as geometric constraints [73, 107, 140]. Compared approaches that impose implicit structure, explicit structures not only improve the learnability but also the transparency of the learned representations. However, most approaches to incorporating explicit structure into deep learning models utilize domain-specific rules and constraints that may not generalize to other tasks and environments.

In this thesis, we study how to improve the reliability and transparency of ML systems by imposing (as well as validating) explicit structures in ML models in the form of simple parametric models, which are defined as probability distributions f from the following set:

$$P = \{f(x_1, x_2, \ldots, x_n; \theta) \mid f : R^n \mapsto R, \theta \in \Theta\}$$

In particular, we study the use of models such as univariate and multivariate distributions, as well as multi-armed bandit models, and explore their practical applicability within three types of machine learning systems. We focus on parametric models since they are a more general tool for incorporating assumptions into ML systems compared to the aforementioned domain-specific structures. We show that it is necessary to ensure their validity in the systems being modeled. Our work therefore proposes strategies for incorporating and validating structures based these simple parametric models. In Chapter 2, we show how to build a reliable and transparent ML policy for a warehouse-scale storage system by using log-normal distributions to model the variability of object lifetimes. In Chapter 3, we outline an experimental framework that we use to empirically test the stationarity assumptions of bandit-based recommendation systems. Finally, in Chapter 4, we describe our proposed work on investigating the (high-dimensional gaussian) latent space geometry of Transformer variational auto-encoders (VAEs) for text. We propose further investigating its isotropic (i.e. sphere-like) qualities and, with this understanding, developing improved methods for transforming sentences in latent space. We outline each of our case studies below.

1.1 Case Study 1: Modeling Variability in Warehouse-scale Storage Systems

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 [], 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 previously discussed that 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 borrowed from survival analysis. 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: 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. Some kinds of 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 validating a common assumption made by bandit-based recommendation systems – that human preferences remain static regardless of the content consumed – holds true in practice. 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. CYCLE is a sequence where the system deterministically cycles through one instance of each of the K categories, i.e. $1, 2, \dots, K, 1, \dots, K$. On the other hand, REPEAT is a sequence where the system enumerates through all instances of category k before then moving on to category $k + 1$, i.e. $1, 1, \dots, K, K, \dots, K$. 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.

1.3 Case Study 3: Geometrically Structured Models for Language Generation Systems.

In our final chapter, we present our proposed work. Previously, we have investigated the use of structure in two types of ML-driven decision-making systems. For our proposed work, we turn our attention towards systems for language generation; in particular, we focus on auto-encoder models for text. Auto-encoders are among the earliest models used for unsupervised learning, and obtain a mapping between the original data space and a continuous latent vector space using a reconstruction-based learning objective. In the context of language generation, recent work has explored using auto-encoders for controlling language generation by manipulating the representations of text in the learned latent space. These works largely do so by leveraging the apparent linear structure of the latent space. This is done by applying offset vectors (e.g. obtained from paired text examples) that can transform high-level attributes of the text such as tense and sentiment. However, the vast majority of work does not attempt to examine or characterize the nature of this structure or the reasons for and extent of its existence. For instance, using Variational Auto-encoders (VAEs) to impose a Gaussian prior (and posterior) on the latent space with the machinery of variational inference is assumed to provide benefits such as "smoothness", "compactness", and isotropy. Our work therefore aims to tackle the following set of questions:

- What kind of geometric structure exists in the latent space of auto-encoders, and how does training a VAE affect that structure?
- How does the structure of the latent space affect the ability to conduct sentence transformations?
- Are there other approaches to transforming sentences that better utilize that structure?

As an initial step towards answering the above questions, we discuss our preliminary findings. Our first finding is that the transformation ability of Transformer auto-encoders are most sensitive to model initialization and training method (e.g. using a VAE) but not to other factors such as model architecture. We use these observations as the basis for the model used for our studies. Our second finding is that the latent space induced by the VAE is by far more isotropic (i.e. spherical) than that of vanilla AEs. We quantify this using several metrics computed using expected pairwise cosine distance, PCA, and KNN-based'distance. We describe these metrics in greater detail in Chapter 4.

Based on these preliminary findings, we are motivated to explore the following directions in our proposed work. First, we aim to explore whether there are lightweight transformations that take better advantage of the particular spherical geometry of the VAE latent space. For example, the existence of spherical geometry suggests that linear traversals through the space may be better modeled as rotations. We examine two generalizations of the standard offset vector-based transformation, including (a) computing a (per-example) weighted linear offset based on the nearest neighbors, and (b) using input-output pairs to supervise a rigid body transformation consisting of rotation and translation. Second, we explore performing additional analyses to probe the semantic sensitivity of the learned representations with respect to small perturbations in the latent space.

Chapter 2

Modeling Variability in Warehouse-scale Storage Systems

2.1 Introduction

Modern data centers contain a myriad of different storage systems and services, from distributed file systems [52, 64, 138, 156], to in-memory caching services [46] and databases [33, 143]. These services typically operate behind an RPC abstraction and are accessed by workloads that are composed of interconnected services communicating through RPCs [50].

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 [13, 69], and block allocators reduce fragmentation by colocating allocations of comparable lifetime [80].

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 [74], 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 [8, 139] and resource managers [34, 114] 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 [112]. 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

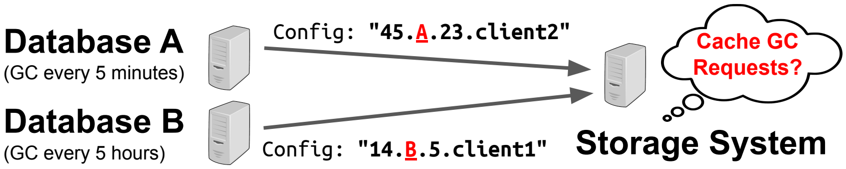


Figure 2.1: Distributed tracing tags contain unstructured application information that can be leveraged to make storage predictions.

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 2.1). 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 2.2) and show that strong predictive performance relies on leveraging the latent structure of unstructured distributed traces (Section 2.3). 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 2.4) 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).

2.2 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 [156] or D file servers [131]), running at the file system level (e.g., HDFS [138]) or storing structured data (e.g., Bigtable [26]). 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 [114], 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 [8, 139], 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.

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. [51] 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 [34, 114] and collaborative filtering [37] have been shown to work well for cluster scheduling while the aforementioned work on distributed traces relies on LSTM neural networks [62]. It is important to

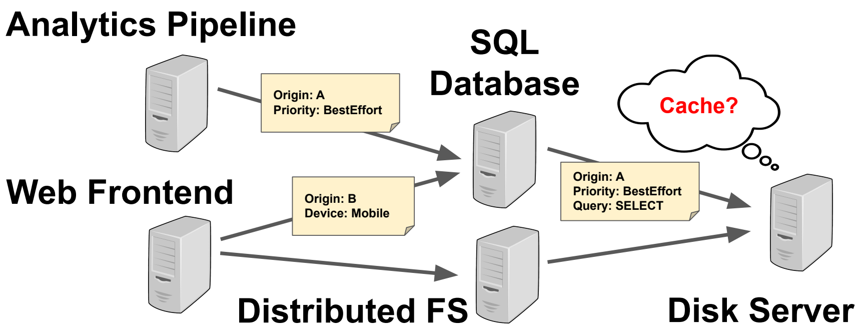


Figure 2.2: Census tags are free-form key-value pairs that are added by services and propagated with subsequent requests.

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

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 [83, 98, 142]; 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 [149], such as GPT-2 [119] and BERT [40].

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.

2.3 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.

2.3.1 Distributed Traces and OpenCensus

Data center applications are composed of services that communicate via message passing [50]. 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 [112] is a distributed tracing library that provides insights into such systems. It tags requests as they travel through the system (Figure 2.2). 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 2.1 (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.

2.3.2 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 [13, 69], 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 [80]. 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 [26, 113]. 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 2.6.4).

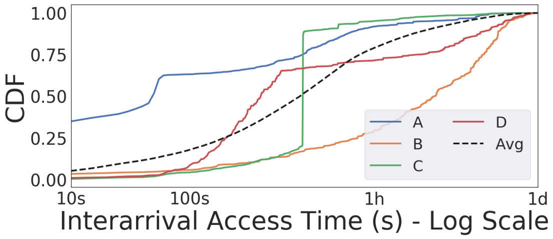
This list is not exhaustive. Other tasks that are not explored in this paper include 1) Resource demand forecasting when deploying a new mix of workloads (e.g., when bringing up a new cluster of machines), by recording a small number of samples characterizing the mix of workloads and then using a model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

2.3.3 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 [131] 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.

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.



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

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)*.

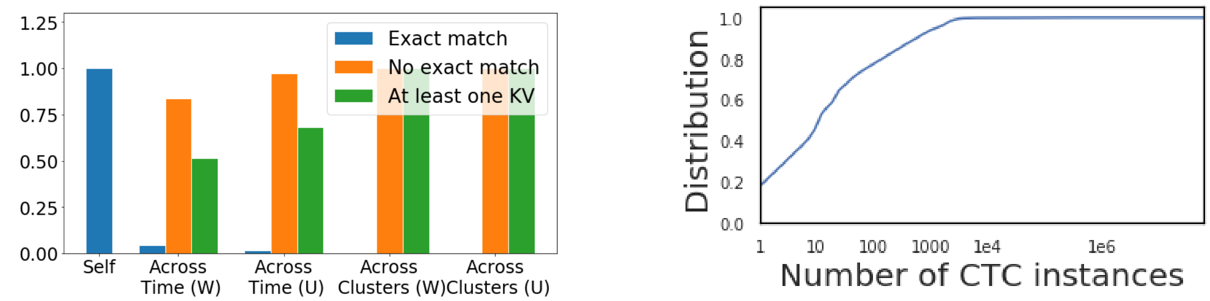
Entropy Analysis. To measure the predictive ability of Census Tags, we look at the distribution of values for each of the tasks we aim to predict (e.g., interarrival times) and compare this distribution to the same distribution conditioned on different values of particular Census Tags. Figure 2.3(a) 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 2.3(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.
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 2.4(a)). We find that 1) only a small fraction of requests have a CTC that occurs exactly in the original trace, but 2)



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

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

Figure 2.4: Transferability (a) and cardinality (b) of Census tags.

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 2.4(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 [77, 137], where tokens are often highly specific to the context in which they appear.

2.3.4 Distribution-based Storage Predictions

Intuitively, we would expect a predictor for the storage prediction tasks from Section 2.3.2 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. [114] 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 [13]: 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 2.4(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 [44]. 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 [111]. 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 2.5 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 .

2.3.5 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 2.3(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 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.

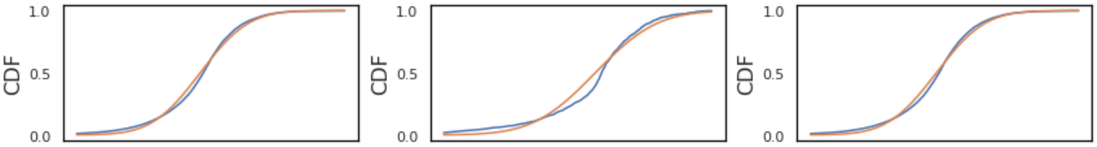


Figure 2.5: Fitting lognormal distributions to CTC CDFs.

2.4 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, $Y = (\mu_Y, \sigma_Y)$.

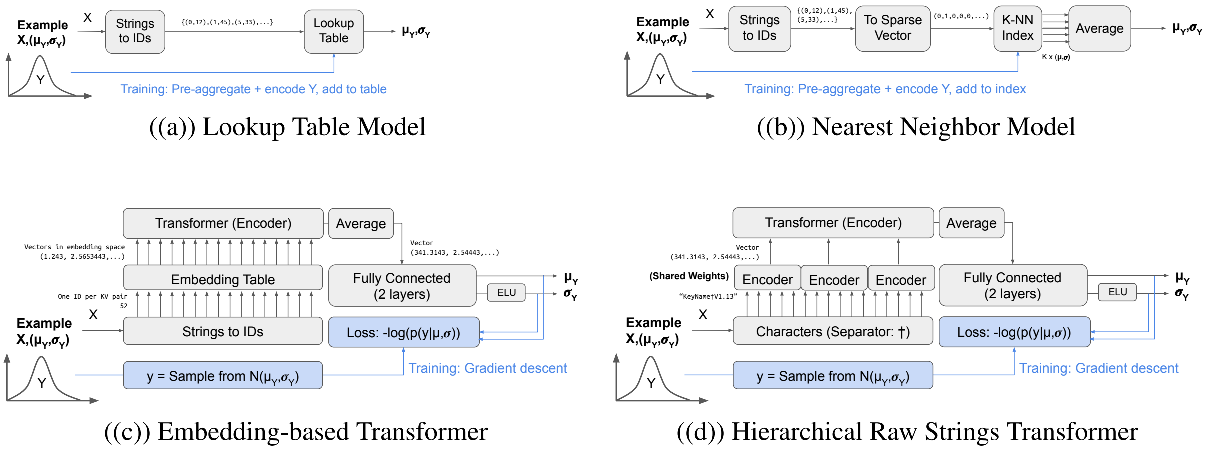
2.4.1 Lookup Table Model

The simplest prediction approach is a lookup table (Figure 2.6(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 2.3.3, 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 [126], which is an equivalent but potentially more compact representation.

2.4.2 K-Nearest Neighbor Model

Improving upon how the lookup table handles *unseen* CTCs, the k-nearest neighbor approach (Figure 2.6(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



((a)) Lookup Table Model

((b)) Nearest Neighbor Model

((c)) Embedding-based Transformer

((d)) Hierarchical Raw Strings Transformer

Figure 2.6: The different models (blue is training).

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 [34, 114]. 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.

2.4.3 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 raw strings. Specifically, we use a Transformer [149] 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* [106], 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 2.6(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 2.6(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

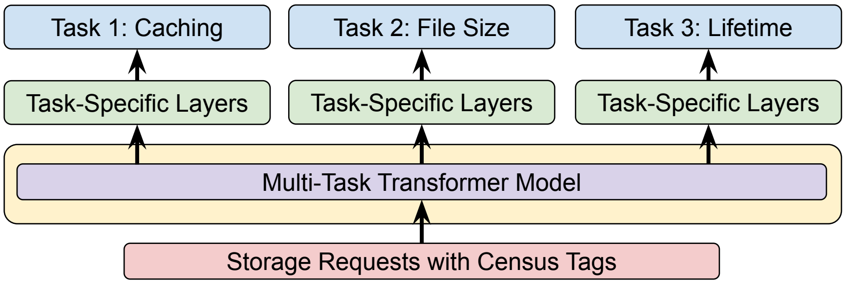


Figure 2.7: Using the Transformer in multi-task learning.

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 label y came from this distribution:

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

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 2.7). 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.

2.5 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 [25] 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 [61]. 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.

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 2.2: 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.

Transformer. We implement our Transformer models in TensorFlow [1] and run both training and evaluation on TPUs [71], using the Tensor2Tensor library [150]. We use the following hyperparameters: {`num_hidden units=64, num_hidden_layers=2, num_heads=4`} 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.

2.6 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.

2.6.1 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 2.2). 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.

2.6.2 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.

2.6.3 Production Traces

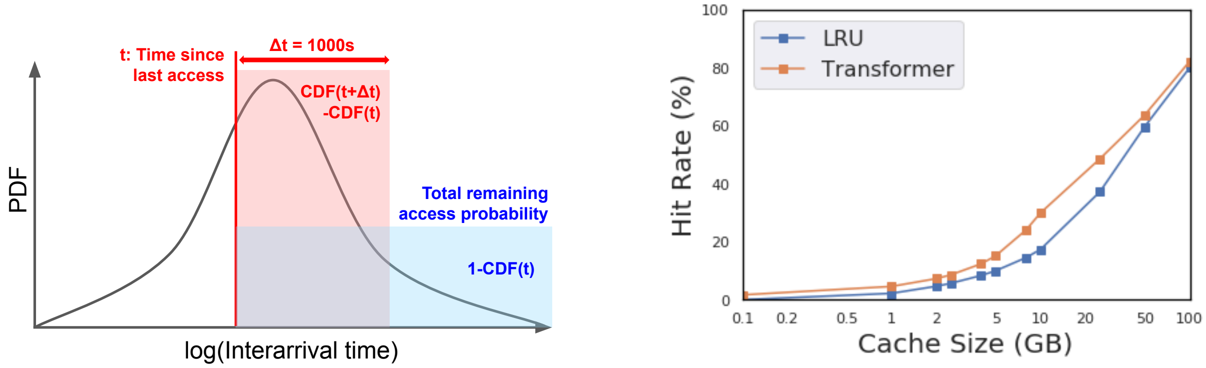
We now evaluate our models on real production traces. Our evaluation consists of two main components: evaluating model generalization error, and demonstrating end-to-end improvements in simulation.

As discussed in Section 2.3.3, 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.

2.6.4 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



((a)) Calculating Utility

((b)) Cache Hit Rate

Figure 2.8: Using predictions in caching.

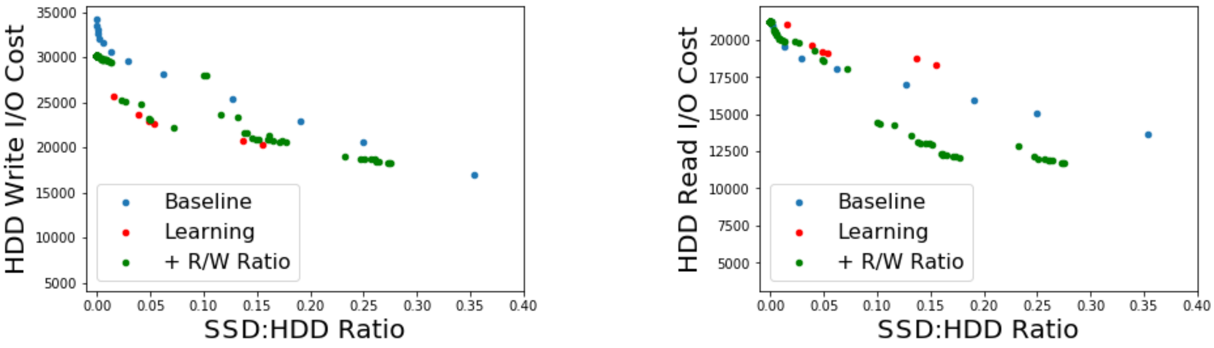


Figure 2.9: Using predictions in SSD/HDD tiering.

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 [4], 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 probability-based replacement policies such as EVA [13] or reuse interval prediction [69]. 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. [4]. 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 2.8(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 2.8(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 [4] 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 2.9). 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 2.3.2) and prefer placing read-heavy files on SSD. This keeps the write savings while also improving reads.

2.7 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 [47]) or WordPiece [157], 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.

2.8 Conclusion

Our results demonstrate that information in distributed traces can be used to make predictions in data center storage services. We demonstrated three models – a lookup table, k-NN and neural-network based approach – that, when combined, extract increasing amounts of information from Census tags and significantly improve storage systems.

Chapter 3

Validating Stationarity in Bandit-based Recommendation Systems

3.1 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 [54]. While these systems are prevalent in real-world applications, we often observe misalignment between their behavior and human preferences [68]. 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, 54], including latent factor models such as matrix factorization [86] and deep learning approaches that are designed to predict various heuristically chosen targets [155] (e.g., purchases, ratings, reviews, watch time, or clicks [15, 103]). 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) [70, 145].

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 [87, 141]. 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 [87]. 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 3.3.1.

K-armed bandits [9, and references therein], contextual bandits [97, 104, 105, and references therein], and Markov decision processes [27, 28, 29, 30, 133, 153, 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) [123], 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 [48]. 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 [58]. 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 [35], 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 3.4). 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 [27, 28, 29, 30, 104, 105, 133, 153] 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 [104, 105], 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 [27, 28, 29, 30, 133, 153], 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 3.3.3). 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 3.4). As an illustration of the flexibility of our experimental framework, we have inspected different bandit algorithms in terms of user enjoyment and attentiveness (Section 3.5). We discuss the limitation of our study in Section 3.6.

3.2 Related Work

The study of evolving preferences has a long history, addressed by such diverse areas as psychology [32, 48], economics [60, 118], marketing [147], operations research [11], philosophy [7, 99], and recommender systems [76, 90, 120]. 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. [93], Seznec et al. [132] model rewards as monotonic functions of the number of pulls. By contrast, Basu et al. [10], Cella and Cesa-Bianchi [24], Kleinberg and Immorlica [85] consider the reward to be a function of the time elapsed since the last pull of the corresponding arm. In Mintz et al. [108], rewards are context-dependent, where the contexts are updated based on known deterministic dynamics. Finally, Leqi et al. [92] 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

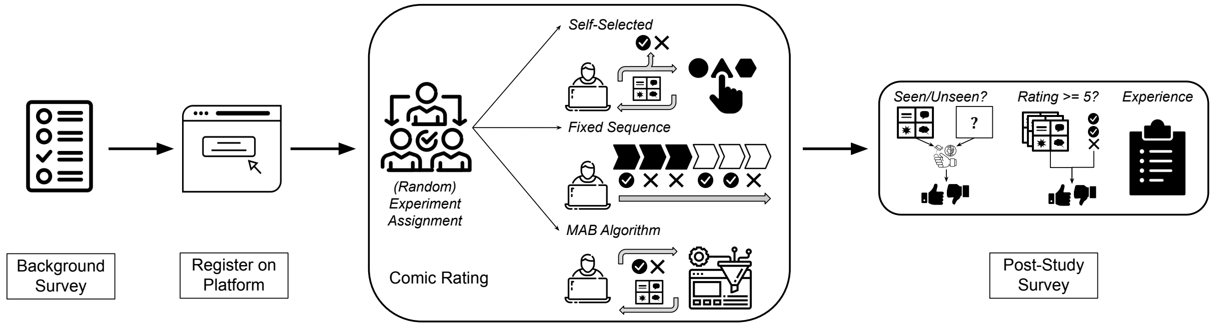
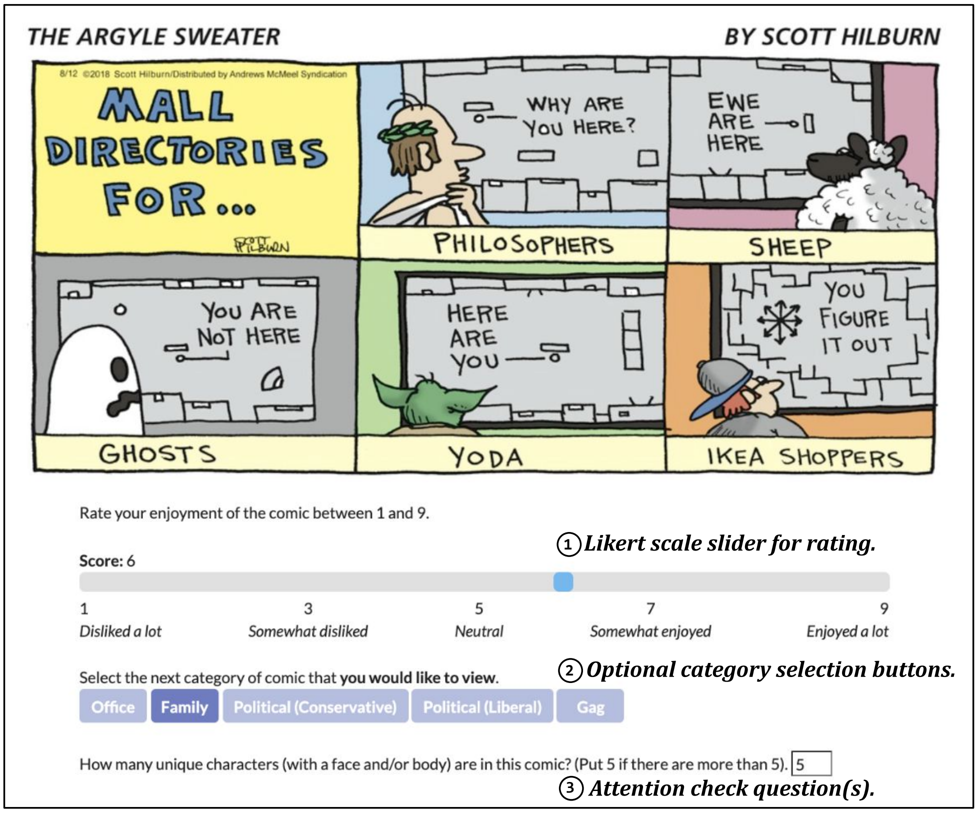


Figure 3.1: Overview of the experimental protocol. Participants first complete a background survey and then register their MTurk ID on our platform to get randomly assigned (without their direct knowledge) one of the following types of experimental setups: Self-selected, one of the fixed sequences, or 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 [27, 28, 29, 30, 104, 105, 133, 153].

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 [91, 127]. 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., [63]) 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, 89, 122]. 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.



Rate your enjoyment of the comic between 1 and 9.

Score: 6

①*Likert scale slider for rating.*

Select the next category of comic that **you would like to view.**

②*Optional category selection buttons.*

How many unique characters (with a face and/or body) are in this comic? (Put 5 if there are more than 5). 5

③*Attention check question(s).*

Figure 3.2: 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.

3.3 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 3.3.1). We then provide reasoning on why we choose comics as the reommendation domain and selection criteria for the comics used for recommendations (Section 3.3.2). Finally, we discuss our experimental framework (Section 3.3.3).

3.3.1 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 [87], 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 3.4, 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 $\frac{1}{t}$ [6]. 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 3.4.

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

3.3.2 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 [129]. In such cases, the effects of recommendations towards the participants' preferences may be minimal.

We collected comics from 5 comic series on GoComics [53]. 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 3.6. In Section 3.3.3, we discuss our experimental protocol and platform.

3.3.3 Experimental protocol and platform

We first outline our experimental protocol, which consists of the following steps (Figure 3.1):

1. *Background survey (initial filtering)*: We ask the study participants to complete a brief background survey (Appendix **??** in the supplementary material). 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 3.3.1. 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 **??** (Appendix **??** in the supplementary material).

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 [35] 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

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

Table 3.1: Number of participants for each algorithm. The description for Self-selected, UCB, TS, ETC and ε-Greedy are in Section 3.3.1. CYCLE and REPEAT are defined in Section 3.4.

on the scale. In addition, in the Self-selected setting where the participants choose the genre of 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 plan to open source our code and data. For more details about the experimental platform and MTurk-related implementation details, we refer the readers to Appendix **??** in the supplementary material.

3.3.4 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 Appendix **??**, 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 3.1 shows the number of participants for each experimental setup.

3.4 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 [87, 123, 141], 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:

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

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 [45] 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 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 [45]: 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 Appendix **??**) 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 3.2). 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 [11, 48]. 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 [129]. For a more comprehensive discussion on preference formation, we refer the readers to Becker [12].

		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 3.2: 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.

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 3.2), 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 3.2, 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 3.3). By looking at the reward trajectories (and the lines fitted through the reward trajectories) over the entire recommendation sequence (Figure 3.3), 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 3.4). 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 [49]. Precisely understanding the causes of this distinction between heavy and light readers is of future interest.

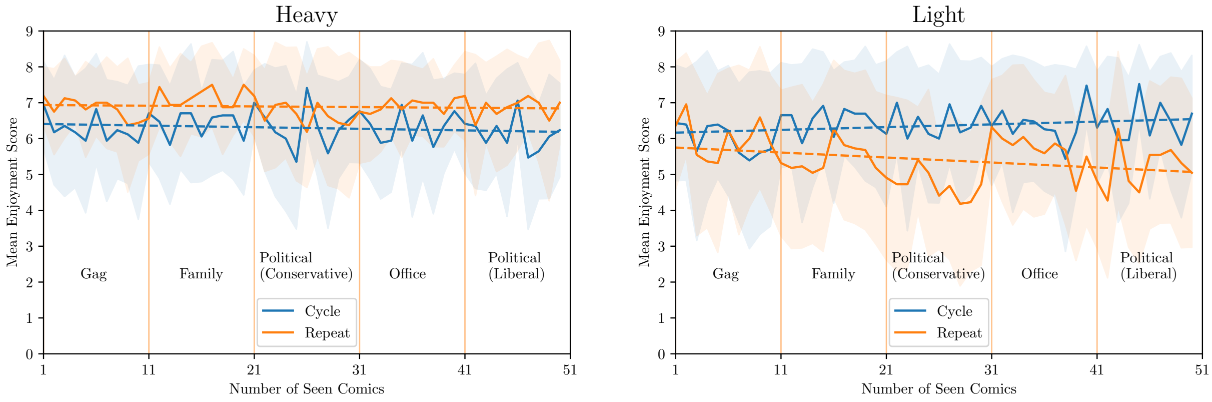


Figure 3.3: The reward at each time step of the CYCLE and REPEAT recommendation sequence averaged across heavy and light readers, respectively. The error bars indicate one standard deviation from the mean. For the REPEAT sequence, we highlight when the arm switches using the vertical lines and add the 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.

	Self-selected	UCB	TS	ETC	ε-Greedy	CYCLE	REPEAT
Cumulative reward	319.36 $[205, 434]$	323.70 $[223, 424]$	312.37 $[204, 420]$	324.49 $[209, 440]$	307.59 $[205, 411]$	316.5 $[216, 417]$	301.63 $[201, 427]$
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 3.3: 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.

3.5 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 3.4 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.

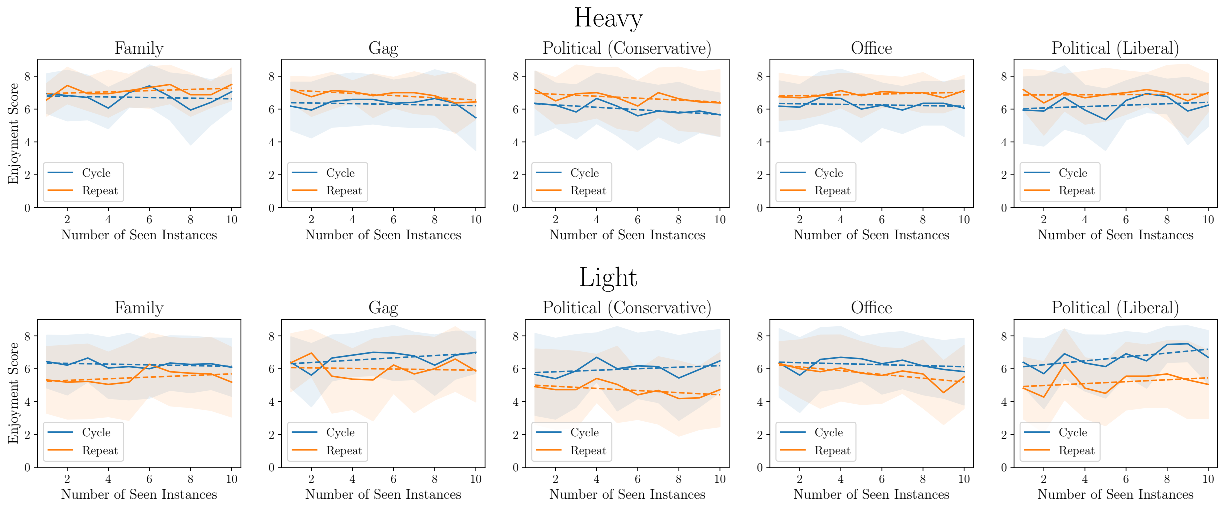


Figure 3.4: Each plot shows the reward collected for a particular arm at each arm pull under the CYCLE and REPEAT recommendation sequences. The error bars indicate one standard deviation from the mean. The reward trajectories are averaged across heavy and light readers, respectively. The blue and orange dotted lines are fitted through the reward trajectories for each arm under CYCLE and REPEAT, respectively.

3.5.1 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 [116]. 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 [87]. 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 3.3, 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 3.3 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

[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 3.4: Average correctness of the two types of memory questions for each algorithm.

read or selecting comics to read on your own?" The third row of Table 3.3 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 3.3). 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 have more autonomy. We provide additional analyses on comparing these algorithms in terms of participant's enjoyment in Appendix **??**.

3.5.2 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 [42]. There is no consensus on defining and measuring mindfulness of an experience [57]. Some of the existing research uses self-reported mindfulness as a measurement [55]. 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 3.4 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 3.4 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 3.4 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 **??**.

3.6 Conclusions and Future Work

Our work provides a general experimental framework and toolkit to understand assumptions on human preferences in a bandit setup. It also allows one to perform field tests of different bandit algorithms that interact with humans. Using these tools, we build a publicly available dataset that contains trajectories of recommendations and ratings of them given by multiple users in a bandit setting. The collected data has been used to check the validity of a core assumption on human preference in MABs literature—that the reward distributions (corresponding to user preferences) are fixed. We show that even in a short time period like our bandit recommendation setup, such dynamical preference exists. Further, our observations on the difference between the light and heavy user in their preference dynamics suggest the need of having a more granular understanding of human preferences. As an illustrative usage of our experimental framework, we have explored the study participants' preferences towards selecting content to read on their own and being recommended content to read. As we have discussed above, these findings are exploratory in nature with the goal of showcasing different usages of our experimental framework; thus, they should not be interpreted as definitive answers. In our exploratory analysis, we observe that an algorithm achieving the highest cumulative reward does not necessarily imply that it will achieve the highest hindsight satisfaction.

At a higher level, our work fits in the broader picture of understanding the validity of assumptions that machine learning systems (and in our case, recommender systems) rely on. Although all machine learning models are built upon some simplifying assumptions of the world, some of these assumptions oversimplify the world to the extent that they lose the core characterizations of the problem we are interested in. In our case, the assumptions we want to understand are centered around user preferences. We have identified that assumptions on the temporal stability of preferences used in traditional MABs literature are oversimplifications. The balance between identifying simplifications that are helpful for building learning systems and avoiding oversimplifications that discard core characteristics of the problem is difficult. As put by the famous statistician George Box, "all models are wrong, but some are useful" [21]. Our goal for developing the experimental toolkit and conducting the human subjects study is to provide ways for identifying assumptions on user preferences that are useful for developing MABs algorithms in recommender systems. Below we discuss the limitations and future directions of our work.

3.6.1 Limitations

We discuss several limitations of our study. First, the sizes of the mean reward differences of each arm reported in Section 3.4 are within 1.5 point on the 9-point Likert scale, which may be considered small. This is due to many reasons. For one, the enjoyment score (the reward) provided by each participant is subjective and may have high variance due to this subjectivity. Though a difference may be considered to be strong in a within-subjective study, it may be thought of as small in a between-subject study (the type of study in our case). However, we note that our results obtained using the permutation test show that reward distributions are indeed not fixed over time for multiple arms in the K-armed bandit recommendation setup. Second, each arm represents a comic series. Though we have selected the comics from each series in terms of their quality (the number of likes that the comics receive), there may still be heterogeneity among the selected comics belonging to the same series, and it is up to discussion on whether one should consider these comics to belong to the same arm. Thirdly, many quantities (e.g., enjoyment/satisfaction, mindfulness/attentiveness) we want to measure are less well-defined. Our way of measuring them is from a particular angle, and may not be widely applicable. Fourthly, the study domain is chosen to be comics due to reasons including the short consumption time of a comic and the study requires the participants to read 50 comics. Although all of our experiments are completed within 30 minutes, it is uncommon in real-world settings for people to read 50 comics at a time and thus may introduce uncontrolled boredom effects. Fifthly, in our experiments, we compare the user preferences towards each arm using two fixed sequences. One may also utilize a purely randomized sequence as a baseline to better understand user preferences. Finally, due to resource constraints (e.g., funding limits for conducting the study and computational limits on the number of simultaneous experiment instances we can host on our server), we recruited 360 participants (with 316 of them passing the attention checks) for our study. Compared to industry-scale experiments, the number of participants in our study is on the lower end. It is also worth mentioning that our implementations of the bandit algorithms ensure that the comic recommendations only depend on the user's own history, which is not a common practice for recommender systems on existing platforms. In practice, one may utilize other users' interaction histories to warm start these bandit algorithms. Though there are these limitations, we would like to emphasize that our work makes a substantive step towards understanding the applicability of traditional assumptions on user preferences in MABs literature.

3.6.2 Future Work

There are multiple future directions for our work. Our findings on the existence of evolving preferences in a K-armed bandits recommendation setting suggest that in order to study the decision-theoretic nature of recommender systems using the MAB framework, one must account for such preference dynamics. The need for learning algorithms (oftentimes reinforcement learning algorithms) that deal with the impact of recommendations on user preferences have also been proposed in recent works [27, 28, 29, 30, 67, 104, 105, 133, 153, 161]. An important building block for this line of research is to have better modeling of human preference dynamics. Our experimental framework and toolkit can provide more grounding and accessibility for research on it. As an example, our observation that heavy and light comic readers have different preference dynamics can be further investigated using our experimental framework, advancing our understanding on evolving preferences in a more granular way. More broadly, our toolkit can be used for: (i) collecting data using fixed or randomized recommendation sequences or bandit algorithms to identify and estimate preference dynamics; and (ii) conducting field tests of bandit algorithms designed to address evolving preferences.

Our exploratory data analysis on the performance of different algorithms suggests that the human interactants of the bandit algorithms may care about other aspects of their experience in addition to cumulative rewards. For example, Self-selected has a higher percentage of satisfied participants in hindsight compared to UCB, though UCB has a higher average cumulative reward. This suggests that besides traditional performance metrics used to analyze and develop these bandit algorithms, we should consider a broader set of objectives and metrics when studying these problems. On a related note, given that we want our algorithm to account for evolving preferences, when regret (the difference between the expected cumulative reward obtained by a proposed policy and an oracle) is used as the performance metric, the oracle should be chosen to be adaptive instead of the best-fixed arm considered in many MABs (including contextual bandits) literature.

Chapter 4

Proposed Work: Geometrically Structured Models for Language Generation Systems

4.1 Introduction

Pre-trained language models (PLMs) have significantly advanced the state-of-the-art for text generation tasks such as summarization, question answering, and style transfer. Modern PLMs— based on the Transformer architecture [149]— have shown a remarkable ability to generate lengthy, coherent, and diverse texts [22, 94, 119, 121], which can be attributed to their massive scale and the enormous amount of pre-training they undergo on web-scale data. As the language generation capabilities of these models have become powerful, increased work has gone into studying how to control their behavior. Prior approaches to control include learned decoding [39, 59], fine-tuning LLMs to target tasks [115], and learning to generate prompts []. Concurrent with the emergence of PLMs, previous works have also opted to train auto-encoders [20, 95, 101, 109, 134, 135] that learn to represent text in a continuous latent space, which enabled a form of controlled generation using geometric operations on latent vector representations.

In this work, we revisit the use of auto-encoders for controlled text generation in the context of modern Transformer-based LMs. We investigate generative models based on variational auto-encoders (VAEs; Kingma and Welling 81) that can tractably induce latent vector representations in accordance to distributional priors that facilitate text transformations. Much previous work in VAEs was conducted using shallow RNN-based architectures. And while there has been some work that has investigated building text VAEs with Transformers [95], we find that most work has focused on quantitative evaluations with less focus on evaluating their sentence transformation capabilities and exploring the properties of their learned latent space.

In this work, we are focused primarily on emergent properties of embeddings, such as their usefulness in facilitating *sentence transformations*, altering a specific aspect while otherwise preserving meaning (Figure 4.1). We investigate the ability of Transformer-based (V)AEs to perform such transformations. Our contributions are as follows: (i) We demonstrate that Transformer (V)AEs can effectively perform various complex sentence transformations through editing latent representations to a far greater extent than existing published methods. (ii) We thoroughly evaluate the impact of VAE training, model architecture, and initialization on sentence

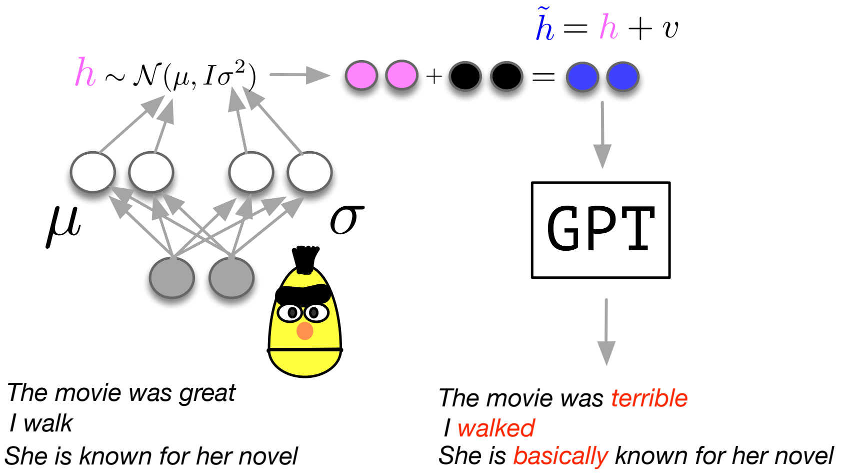

$\tilde{h} = h + v$
$h \sim \mathcal{N}(\mu, I\sigma^2)$
+
=
μ
σ
GPT
The movie was great
I walk
She is known for her novel
The movie was terrible
I walked
She is basically known for her novel

Figure 4.1: Variational auto-encoders (VAEs) that integrate modern pre-trained transformer components can be used to perform various transformations of inputs (e.g., changing tense) reasonably reliably by applying linear operations to latent vectors.

analogy performance. (iii) We conduct in-depth analyses that reveal interesting characteristics of the Transformer VAE and its latent space, such as the ability to compose sentence transformations, the existence of implicit gender-occupation associations, and the isotropy of the learned representations. To our knowledge, this is the first demonstration that these models are capable of performing such transformations on long sentences. We hope that our results can inform future work on Transformer (V)AEs.

4.2 Background and Preliminaries

Pre-trained Language Models. Pre-trained LLMs have led to substantial improvements on natural language benchmarks and generation tasks. These models are typically trained in an unsupervised fashion on large quantities of data. Typically, the pre-training objective consists of predicting text given its relevant context (such as their prefix or surrounding text spans.)

Word embeddings. Word2Vec [106] 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. In particular, the authors demonstrated a simple technique for "analogy-style" transformations (of the form $A : B :: C : D$) using algebraic vector manipulation Examples from Mikolov et al. [106] include semantic transformations, (e.g., *man* : *woman* :: *king* : *queen*) and syntactic transformations (e.g., *big* : *bigger* :: *small* : *smaller*).

Text auto-encoders. More recent work on inducing fixed-length representations of text has used neural architectures to pack sequences of words (e.g., sentences) into vectors [20, 84, 88, 96]. For example, Bowman et al. [19] evaluated fixed-length vector models of sentences using modern auto-encoders. Their work focused in part on addressing learning challenges, and on performing exploratory analyses such as interpolation. Follow-up efforts have proposed improved model architectures [158], complex posterior distributions [82, 136], and optimization methods [] and mostly been assessed quantitatively.

There have been several works that have investigated algebraic "analogy-style" transformations that use auto-encoders and other models to encoding sentences as vectors. Shen et al. [135] trained LSTM auto-encoders to reconstruct corrupted and adversarially perturbed inputs. Montero et al. [109] proposed Autobot, which uses a shallow decoder to reconstruct sentences encoded by a frozen Transformer LM. Subramani et al. [144] proposed directly extracting sentence vector representations in the activation space of Transformer LMs. Another (complementary) line of work learns the sentence transformations in the latent space of vector-based models [101]. All of these works primarily focus on tense and sentiment transformation on sentences from the Yelp dataset, which are comparatively shorter than sentences from IMDB and Wikipedia.

The work that is most directly related to ours is that of Li et al. [95]. Theirs was the first work to learn fixed length vector representations of text using auto-encoders built on Transformer PLMs. They proposed a unified architecture based on a BERT [41] encoder and GPT-2 [119] decoder, and primarily evaluated their model's ability to be fine-tuned to specific downstream classification and generation tasks.

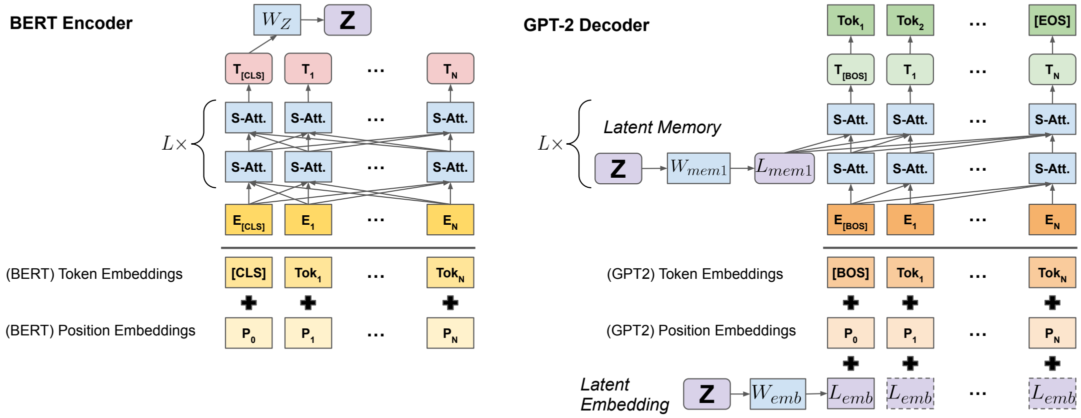


Figure 4.2: The model architecture for our default `memory_and_embeds` configuration.

4.2.1 Variational auto-encoders

VAEs model relationships between data X and latent representations Z using an encoder model $q_\phi(z|x)$ (or *inference network*) that approximates posterior $p(z|x)$ and a decoder model $p_\theta(x|z)$. The inference network defines variational parameters ϕ, also to be estimated. The prior is typically selected to be $\mathcal{N}(0, \mathrm{I})$ and the approximate posterior from a similar family of distributions; e.g., diagonal Gaussians with mean and variance $\mathcal{N}(\mu, \mathrm{I}\vec{\sigma}^2)$.

VAEs are trained by optimizing the Evidence Lower Bound (ELBO) on $p(x)$:

$$\log p(x) \geq E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{\text{KL}}(q_\phi(z|x)||p(z)).$$

The model must balance optimizing the reconstruction loss $\mathcal{L}_r = -E_{q_\phi(z|x)}[\log p_\theta(x|z)]$ against ensuring that $q_\phi(z|x)$ is similar to the prior distribution $p(z)$. In particular, VAEs are prone to *posterior collapse*, where the encoder ignores the input and sets all z equal to the prior. To alleviate this, we examine the whether the VAE training approach used by Li et al. [95] is effective. Their approach combines (1) KL divergence annealing (between 0 and 1 over many epochs) from Bowman et al. [19] with (2) KL divergence dimension-wise thresholding by λ , i.e. optimizing $\mathcal{L}_{\text{KL}} = \sum_{i=1}^{K} \max(\lambda, D_{\text{KL}}(q_\phi(z_i|x)||p(z_i))$, from Kingma et al. [82].

4.3 Transformer auto-encoders

4.3.1 Model Architecture

We use BERT$_{\text{BASE}}$ as the encoder model. The parameters determining the distribution of latent variable z are obtained by applying a linear layer $W_Z \in R^{2d \times h}$ to the last-layer hidden representation of the [CLS] token:

$$\big[E[z]; \ \ \log(\text{Var}(z))\big] = W_Z h_{\texttt{CLS}}.$$

The latent z is then sampled from the distribution $\mathcal{N}(E[z], \text{Var}(z))$ and passed into a GPT-2 decoder.

We use an architecture similar to Li et al. [95] as our primary model, shown in Figure 4.2. This approach introduces additional parameters $W_M \in R^{2Lh \times d}, W_E \in R^{h \times d}$ and passes z into the GPT-2 decoder by (1) adding $h_{\text{EMB}} = W_E z$ to all input embeddings, and (2) using z as external ”memory”, allowing the decoder to attend to key and value vectors $[K; V] = W_{M[2lh:2(l+1)h]} z$ at each layer l. We refer to this architecture as memory_and_embeds. We find that memory_and_embeds performs well, although is not the only architecture that does so. Our architecture ablation results are Section 4.5.

4.3.2 Training details

Training approach. For the AE, we set the KL Divergence weight to zero and train the VAE deterministically by directly passing the mean into the decoder (ignoring the variance). For the VAE, we use a training scheme that combines annealing and the “free bits” model. We first train the VAE for 1 epoch with the weight of the KL divergence term set to 0, then linearly anneal its weight between 0 and 1 for 0.5 epochs, and subsequently train with weight 1. We use the dimension-wise ”free bits” form of KL divergence with $\lambda = 0.5$. More details can be found in the Appendix.

Model initialization and tokenization. We initialize the BERT encoder and GPT-2 decoder with their pretrained LM checkpoints while using their respective tokenizers, i.e. split tokenization with pre-training. We find that initializing auto-encoders from pre-trained checkpoints (while retaining the different tokenizers) results in the best downstream performance. Our initialization ablation results are in Section 4.5.

Data Processing. We train models on a sentence corpus from Wikipedia comprising 98.4 million sentences. Sentences are truncated to 64 tokens, which allows 96.3% of sentences to retain their full length. We split the corpus into 101 contiguous partitions, and use the first 100 partitions for training and the final split for evaluation.

4.4 Transforming Sentences

In this section we show how to transform sentences in targeted ways via manipulations in the induced latent space. Specifically, we explore approaches to exploiting light supervision (1-10 input/output pairs) to learn to perform particular transformations via operations on latent representations.

Model	**Feature**	**Acc.**	$E[\cos(x,y)]$	**KNN**	**PCA**		
				Overlap	0.8	0.95	0.99
AE	μ	91.1	0.7849	0.375	341	467	547
VAE	μ	58.4	0.9613	0.217	256	334	420
VAE	$\mu_{z\mathrm{NORM}}$	61.6	0.9613	0.217	256	334	420
VAE	z	35.8	0.9820	0.143	561	714	757

Table 4.1: Exact reconstruction accuracy and various approaches to quantifying isotropy using various VAE latent representations on the $\texttt{Wiki}_{\text{TEST}}$ dataset. The sampling results are computed using $n = 10$ samples.

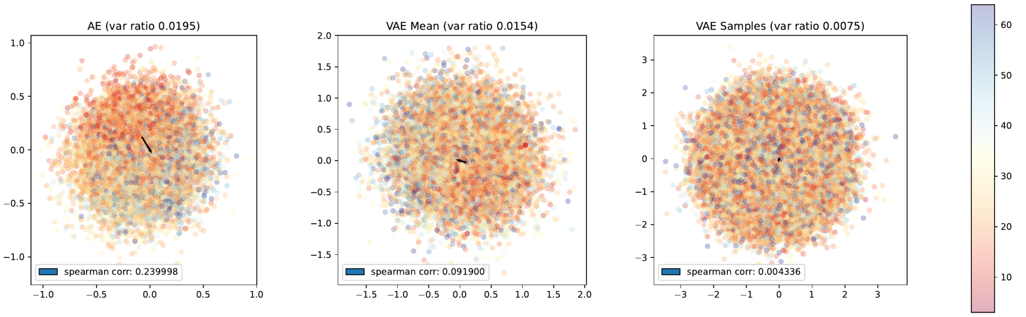


Figure 4.3: The first two principle components for the AE and VAE vector representations of $\texttt{Wiki}_{\text{TEST}}$, colored by the length (in tokens using the BERT tokenizer.) We plot the direction that maximizes the Spearman correlation between the projection along that direction and the sentence lengths. We observe some positive correlation for both the AE and VAE means, though it is most apparent in the AE means.

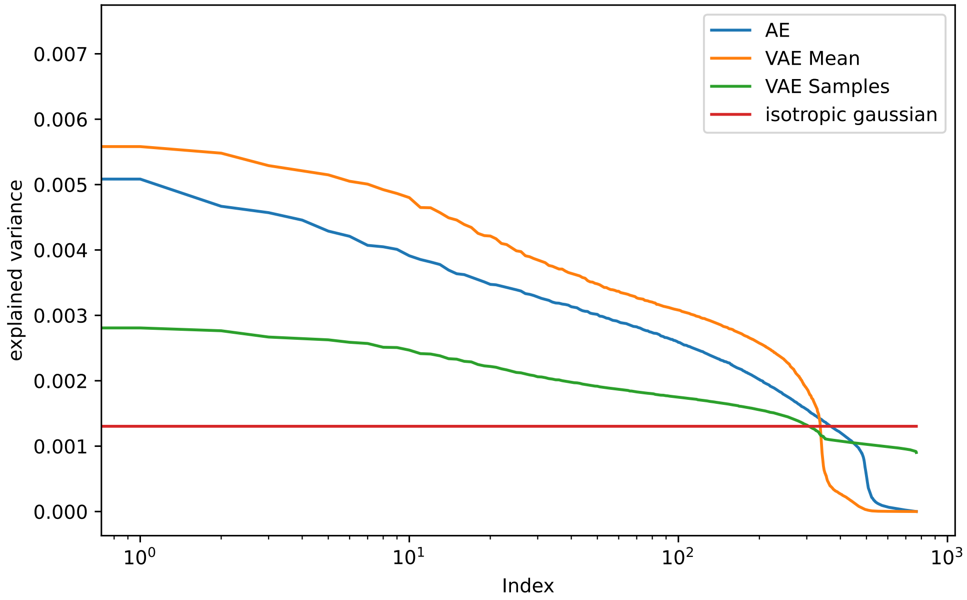


Figure 4.4: The explained variance of the principle components for the AE and VAE vectors on $\texttt{Wiki}_{\text{TEST}}$.

4.4.1 Analogy-style transformations

We apply "analogy-style" transformations to sentences as follows. Given input-output pairs, we compute differences between their latent representations, and then average these difference vectors to obtain a mean offset. Note that this is the translation vector t that minimizes the squared error between two sets of points X and Y:

$$\hat{t} = \underset{t}{\operatorname{argmin}} \sum_{i=1}^{N} \|x_i + t - y_i\|_2^2$$

where x_i and y_i denote the i^{th} row of matrices containing the original and transformed sentence representations X and Y. We then perform the transformation by applying the resultant offset vector (and its extrapolations) and then decoding the result.

In addition to applying the mean vector, we also experiment with two other approaches to computing linear offset vectors. In particular, these methods allow us to make use of *paired* supervision.[1] The approaches we explore involve computing a weighted sum of the offset vectors for each input example x_j, i.e.

$$\hat{t}_{x_j} = \frac{1}{\sum_{i=1}^{N} w_i^{(j)}} \left(\sum_{i=1}^{N} w_i^{(j)} * (y_i - x_i) \right)$$

[1] In contrast, using the mean offset vector is equivalent to taking the difference between the means of the original and transformed sentences, which can be accomplished with unpaired sentences.

where the w_i's are functions of x_j, X, and Y. We first explore an approach based on averaging only offsets that correspond to x_i's which are the K nearest neighbors of x_j. The weights in this scheme are given by $w_i^{(j)} = \mathbf{1}[x_k \in \mathrm{NN}(x_j, K)]$ where the nearest neighbors are defined in terms of either euclidean or cosine distance. The motivation behind using a nearest neighbors-based approach is that the average offset vector may not be able to fully capture the transformation – e.g. when the manifold of the latent space is highly non-linear – but that the space may be locally linear in the sense that vectors x_i near x_j have similar offset vectors.

The second approach we explore is our proposed "soft" version of nearest neighbors that weights offset vectors proportional to the cosine similarity of their x_i's from x_j. The weights are computed by taking a softmax over the cosine similarities scaled by a fixed constant $s \geq 1$. The scaling step is necessary since cosine similarity is bounded in the range $[-1, 1]$, and is similar to using a temperature constant to flatten or sharpen logits in softmax. Formally, for $\cos\theta_i^{(j)} = x_i^x{}_j / (\|x_i\| \|x_j\|)$, the weights are given by

$$w_i^{(j)} = \frac{e^{s\cos\theta_i^{(j)}}}{\sum_{i=1}^N e^{s\cos\theta_i^{(j)}}}$$

This formulation has been used in metric learning [38, 151, 152], typically to encourage large margins between L2 normalized vectors that lie on a hypersphere, but has also seen use in other areas such as out-of-distribution detection [146]. We conduct analyses in Section 4.4.3 to study how latent space cosine similarity relates to offset similarity, and how sensitive the weighting function is to the choice of s.

We also consider alternative methods for performing sentence transformations that leverage the isotropy of the VAE latent space; we explore a class of methods that makes use of *paired* supervision, in contrast to previous methods (such as computing the mean vector.) Since the VAE latent space mostly contains vectors with similar norms, its geometry is generally spherical. Intuitively, applying an offset t in spherical geometry is analogous to rotation by a matrix R. We first explore learning a rotation matrix that minimizes the squared euclidean distance between the transformed inputs and their corresponding outputs,

$$\hat{R} = \underset{R}{\operatorname{argmin}} \sum_{i=1}^{N} ||Rx_i - y_i||_2^2$$
$$\text{where } R^T R = I$$

(This is known as the "Orthogonal Procrustes problem"; Hurley and Cattell 65, Schönemann 130.) The optimal solution $\hat{R}$ is obtained through the SVD of the correlation matrix $USV^T = X^T Y$, and is given by $\hat{R}_{OP} = VU^T$.

We also explore an approach that combines translation and rotation. We learn a rigid affine transformation $A = [R|t]$ by optimizing the following:

$$\hat{R}, \hat{t} = \underset{R,t}{\operatorname{argmin}} \sum_{i=1}^{N} ||Rx_i + t - y_i||_2^2$$

Using the Kabsch-Umeyama algorithm [72, 148], $\hat{R}$ and $\hat{t}$ can be solved in closed form using the SVD of the correlation matrix between the centered data matrices $USV^T = (X - \bar{X})^T (Y - \bar{Y})$

as $\hat{R}_{KU} = VU^T$ and $\hat{t}_{KU} = \bar{Y} - \hat{R}_{KU}\bar{X}$. While this approach offers greater expressivity than translation and rotation alone, in practice it overfits. This is because the number of parameters (on the order of $d(d-1)/2$) is much greater than the amount of supervision we typically use. Ideally we would like to regularize the rotational transformation so that it performs smaller rotations. Low-rank approximations to the correlation matrix can produce rotation matrices that still have large deviations. Instead, we modify the optimization objective to include an additional regularization term $\lambda|R - I|_F^2$. To our knowledge, we are the first to propose (i) using a rigid affine transformation (via the Umeyama algorithm) to transform sentences in latent space, and (ii) regularizing the Umeyama objective to trade off translation and rotation.

4.4.2 Using VAE Representations

Next we discuss how we use the learned representations from VAEs. Training under the VAE objective results in a latent space with particular geometric properties, including misalignment between the means and samples in high dimensions, and *isotropy*—being distributed equally along all dimensions. We investigate these properties and discuss how we account for them when controlling sentence generation.

Using Means and Samples

As noted by Bosc and Vincent [18], previous works on text VAEs typically only use the mean representation μ of the sentence (without explanation or justification) in their analyses. While mean vectors work reasonably for downstream vector-based tasks, they are not representative of what the decoder typically encounters during training. In particular, we observe that the vector norm of μ is substantially smaller than those of its samples z, since adding noise to high-dimensional vectors will tend to move them further from the origin.

We compare the reconstruction accuracy obtained using μ and z to compare their effects on decoding (Table 4.1). We find that using z, however, decreases the reconstruction accuracy somewhat substantially on $\texttt{Wiki}_{\text{TEST}}$. Instead, we explore another approach for extracting more representative vector representations (in terms of vector norm). Specifically, we propose normalizing the mean vector to the average length of the samples, i.e.,

$$\mu_{z\text{NORM}} = \frac{\mu}{\|\mu\|}\left(\frac{1}{K}\sum_{k=1}^{K}\|z_k\|\right)$$

and find that decoding sentences from $\mu_{z\text{NORM}}$ improves reconstruction accuracy.

Isotropy

The posterior distribution induced by the canonical VAE is a diagonal Gaussian with mean μ and covariance $\sigma^2 I$, and their marginal distribution is regularized towards that of a standard Gaussian with mean 0 and unit covariance. The resulting marginal distribution of z is often much more evenly distributed across directions in the vector space compared to representations learned by an AE. Previous work has shown that isotropy can benefit word vectors [5, 110] and sentence vectors [159] due to their "self-normalization" properties. Here, we explore several approaches to quantifying the isotropy from the literature.

We first compute the expected pairwise cosine similarity, i.e. $E_{i\neq j}[\cos(\phi(x_i), \phi(x_j))]$, for featurizer ϕ and data x_k. This metric was previously used to study the contextual embedding space of PLMs [23, 43], and is based off the intuition that samples from strongly isotropic distributions (such as Gaussians) should be orthogonal to each other in expectation. We find this is the case with VAE vectors (particularly the samples) while the AE vectors are more self-similar. Next, following Sablayrolles et al. [125], we quantify the spread of the data by comparing the nearest neighbors with the kth nearest neighbor (using $k = 100$.) Specifically, we compute $p(d(x_i, NN(x_i, 1)) < d(x_j, NN(x_j, k))$, i.e. the probability that the distance between a point and its nearest neighbor is less than that of another point and its kth nearest neighbor. (In an imbalanced space, this quantity approaches 50%.) We refer to this as the KNN overlap, and find that it is 21.7% for the VAE means, compared to 37.5% for the AE. Last, we apply PCA to the latent vectors and report the minimum PCA dimension required to capture $\epsilon = \{80\%, 95\%, 99\%\}$ of the variation in the data [23], i.e. finding the minimum k such that $r_k = \frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{d} \sigma_i} \geq \epsilon$, where σ_i is the i-th largest eigenvalue of the covariance matrix of the data. According to this measure,the VAE sample vectors (although not the mean) are more isotropic than the AE mean vectors.

We additionally plot the first two principle components of the Wiki$_{\text{TEST}}$ dataset representations and find that the AE (and to a lesser extent, VAE) vectors exhibit some skew with respect to sentence length (Figure 4.3). It has been shown that embedding vector norms are correlated with word frequency in both classic and PLM word embeddings Arora et al. [5], Cai et al. [23], Mu and Viswanath [110]. We find that a similar relationship exists for the vector norm of (V)AE sentence embeddings, and observe a clear inverse relationship between the sentence length and vector norm for the AE. The VAE, on the other hand, generally encodes sentences with uniformly sized vectors, except the shortest sentences which are given smaller vectors; however, this difference diminishes when using samples instead means. We hypothesize that this is due to shorter sentences being easier to reconstruct. Hence, the regularization effect of the VAE objective is less effective for these sentences. We posit that the difference in vector norms as a function of sentence length will have an effect on the ease of sentence transformation.

4.4.3 Sentence transformation tasks

We evaluate the extent to which learned representations enable performing these tasks under simple transformations. We evaluate this using a 250-example subset, so that creating test examples using manual editing can be completed with reasonable effort. Following prior work, we use 100 input-output sentence pairs to "supervise" transformations.[2] Below we describe the types of sentence transformations that we perform and how we evaluate them. We investigate 3 types of transformations: sentiment, grammatical, and additive transformations.

Sentiment. We evaluate both negative-to-positive and positive-to-negative sentiment transformations on (1) the Yelp dataset used by prior work Montero et al. [109], Shen et al. [134, 135],

[2] We note that we are able to perform these transformations with far fewer examples ($\leq$10s); see Appendix **??**.

Task	Model	Transformed Sentence
Negative-to-Positive Sentiment	*Original*	*Anyway, the movie is largely boring and based around a bunch of worthless characters.*
	Transformer	Anyway, the movie is largely interesting and based around a bunch of valuable characters.
	LSTM	Anyway, the movie is largely based and greatnna of a toured whatsoever 1A.
	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.
Positive-to-Negative Sentiment	*Original*	*My husband and I enjoyed it so much we bought the VHS and have enjoyed it ever since.*
	Transformer	My husband and I enjoyed it so much we bought the VHS and have enjoyed it ever since.
	LSTM	My husband and I enjoyed it so much bads the VHS bought and we have enjoyed : ever it. awfu
	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.

Table 4.2: Examples of sentiment sentence transformations on IMDB-S across different model types. We report the first correct result obtained through extrapolation.

Subramani et al. [144], and (2) the IMDB-S sentence dataset created by Wang and Culotta [154] from the counterfactually augmented IMDB review dataset of Kaushik et al. [78]. For IMDB, we perform sentiment transformations on the original (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. The success of the transformation is determined using a sentiment classification model trained on a held-out portion of each respective dataset. Example transformations are shown in Table 4.2.

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”. To evaluate transformation success, we check if the corresponding span in the decoded sentence contains the transformed word and verb. Example transformations are shown in Table 4.3.

Adding words and phrases. These transformations involve adding particular words at any location in the sentence and adding phrases at the beginning and end of the sentence. We have the model add unnecessary qualifying words such as ”basically”, ”generally”, and ”essentially” anywhere in the sentence and pre-pend start phrases (such as “Because of this” and “It was apparent that”) as well as end phrases (such as “which was interesting” and “because it is necessary”.) Example transformations are shown in Table **??**.

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.*
	Transformer	He was ordained a religious priest of Bethany Ashram by then Bishop of Kandy H. E Bernad Reinjo O.C.B in 1944.
	LSTM	He was ordained a priest religious of Bethany Ash by then proposed Karam of Bishop H. Endy Berns a UC January MDad. fl from that Poland.
	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.
Singular-to-Plural	*Original*	*The site was praised by Boing Boing, ”The Wall Street Journal”, and ”Business Week”.*
	Transformer	The sites were praised by Boing Boing, ”The Wall Street Journal”, and ”Business Week”.
	LSTM	site are The praiseding by Bobahing, Boites ” ” WallryaconOD, and models : ” 30
	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”.

Table 4.3: 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.

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 *content* via manipulations in latent space. Following prior work, we plot the TS% v.s. Self-BLEU. The points in this curve are produced by extrapolating using fixed coefficients along the analogy vector.

Sentence Transform Vector Analysis

In this section, we analyze the relationship between the sentence vector representations and their offset vectors. We examine how this relationship varies with respect to the inference task and choice of model (AE vs VAE.) First, we plot the cosine distance between the original sentence vectors against the cosine distances between their offset vectors. We find notably stronger correlations between these quantities for the VAE than the AE, and some positive correlation for all tasks besides Present-to-Simple Past. This suggests that there may be some locality in the VAE latent space that would improve the outcome of a neighbor-based scheme for aggregating offset vectors.

We then conduct studies that involve measuring the transformation error on the examples used to supervise the transformation in a manner similar to leave-one-out cross validation, i.e. we perform the transformation without the i^{th} example. This gives us a proxy for the efficacy of these

		Sentiment				Grammar		Additive		
		Pos$_{I}$	Neg$_{I}$	Pos$_{Y}$	Neg$_{Y}$	Past	Plrl	Qual	Beg	End
Baselines	PT-TF AE	**87.3**	87.6	**53.0**	**41.9**	**70.1**	**39.5**	**100**	73.3	49.1
	PT-TF VAE	81.2	76.2	39.2	26.2	51.2	14.5	99.6	**98.3**	**60.2**
	DAAE	-	-	51.8	27.9	-	-	-	-	-
	Autobot	-	-	28.9	13.9	-	-	-	-	-
	Steering Vec.	73.8	60.5	32.3	18.7	25.2	5.3	60.5	25.3	2.6
Small Models	PT-TF AE	**87.3**	**87.6**	**53.0**	**41.9**	**70.1**	**39.5**	**100**	73.3	49.1
	PT-TF VAE	81.2	76.2	39.2	26.2	51.2	14.5	99.6	**98.3**	**60.2**
	LSTM AE	41.4	46.8	19.2	5.5	13.7	0.0	99.2	0.0	0.6
	LSTM VAE	-	-	-	-	-	-	-	-	-
	BiLSTM AE	36.1	30.9	21.4	11.1	16.9	0.0	44.6	0.0	0.6
	BiLSTM VAE	39.2	29.6	28.9	13.9	-	-	72.2	32.1	20.0
	3L-TF AE	54.5	45.0	30.7	17.6	23.0	1.0	87.9	4.9	35.7
	3L TF VAE	58.7	46.8	40.0	24.9	8.4	1.6	59.0	97.2	13.4
Init Ablation	PT-TF AE	**87.3**	**87.6**	**53.0**	**41.9**	**70.1**	**39.5**	**100**	73.3	49.1
	PT-TF VAE	81.2	76.2	39.2	26.2	51.2	14.5	99.6	**98.3**	60.2
	PT-S-TF AE	74.6	75.5	32.2	18.1	43.1	10.1	**100**	58.8	29.5
	PT-S-TF VAE	83.0	82.5	44.5	32.8	50.3	7.5	**100**	69.0	**77.0**
	RI-TF AE	63.0	55.3	34.8	19.8	34.7	10.4	87.8	37.1	56.4
	RI-TF VAE	56.7	50.8	33.8	16.7	39.9	5.4	74.5	65.1	14.0
	RI-S-TF AE	53.3	42.3	31.7	20.2	34.0	1.2	90.5	4.8	37.0
	RI-S-TF VAE	43.2	46.6	37.4	22.0	35.2	2.0	74.0	92.5	55.0
	BI-S-TF AE	80.6	79.9	41.7	26.3	44.6	13.5	**100**	51.7	21.3
	BI-S-TF VAE	70.2	71.9	37.2	21.6	45.0	6.4	**100**	50.5	61.9
Arch Ablation	PT-TF AE	**87.3**	87.6	53.0	41.9	70.1	39.5	**100**	73.3	49.1
	PT-TF VAE	81.2	76.2	39.2	26.2	51.2	14.5	99.6	98.3	**60.2**
	SFT AE	82.3	90.3	53.7	43.5	**79.1**	37.4	**100**	69.6	49.8
	SFT VAE	81.5	77.3	41.6	28.5	13.0	4.8	99.7	84.5	25.4
	AP AE	84.4	**91.1**	**55.3**	**43.6**	72.1	**41.1**	**100**	71.3	39.4
	AP VAE	81.1	82.8	36.6	30.1	47.4	12.8	99.2	**100**	53.2
Scaling	PT-TF AE	87.3	87.6	53.0	41.9	**70.1**	39.5	**100**	73.3	49.1
	MLM AE	**89.7**	91.7	**55.3**	**43.8**	54.4	49.5	**100**	40.9	**61.9**
	LG AE	83.4	**95.3**	52.9	43.5	67.5	48.0	**100**	77.8	31.5
	LG-1024 AE	84.4	91.1	53.9	43.5	60.4	**52.7**	**100**	**91.9**	19.3

Table 4.4: Sentence transformation results for 4 subgroups: baselines, small models, Transformer initialization ablation, and Transformer architecture abalation. All results are shown for Self-BLEU of 30, except for `Past` and `Plrl` which uses a Self-BLEU of 70. '-' indicates that results are only available for Self-BLEU lower than the threshold. The baselines include DAAE [135], Autobot [109], and Steering Vectors [144]. We mainly study the pre-trained (PT) Transformer (TF), but also explore random and BERT-only initialization (RI and BI), shared tokenization (S), substituting the latent vector as the first token (SFT), and average pooling (AP). We also conduct scaling experiments that increase the model size (LG) and latent dimension (1024), and explore a Masked LM AE training scheme.

transformations. We compare the cosine distances and mean squared errors obtained by using the offset of the first nearest neighbor and the mean offset of the first 10 nearest neighbors (out of 100.), and additionally compare against the mean offset of a random subset of 10 examples. We find the first nearest neighbor offset has the highest error, and that using 10 nearest neighbors clearly outperforms 10 random offsets, often achieving comparable error to the leave-one-out mean of all offsets. Lastly, we observe that the cosine-similarity weighted mean of the offset vectors attains lower error than all other approaches, which suggests that this approach holds

Coefficient	Transformed Sentence
Original	*Very bad acting, and a very shallow story.*
$\alpha = 1.0$	Very bad acting, and a very shallow story.
$\alpha = 2.0$	Very bad acting, and a very shallow story.
$\alpha = 4.0$	Very good acting, and a very shallow story.
$\alpha = 6.0$	Very good acting, and a very great story.
$\alpha = 8.0$	Very good, and a very special story.

Table 4.5: Extrapolating the negative-to-positive vector yields more positive adjectives on Transformer AE.

promise.

We note that extrapolating in the direction of a translation vector t leads to sentence representations becoming closer to t (in terms of cosine distance.) However, for large values of α, the extrapolated sentence vector moves further from the manifold, which leads degeneration in the output sentence quality. We therefore explore try to model extrapolation for the VAE by interpolating all sentence towards the offset vector. We find that transforming sentences in this manner is comparable for certain tasks, including ...

4.5 Evaluation

4.5.1 Sentence Transformation Results

We present numerical transformation results in Table 4.4, and next address several research questions.

Are VAEs better than AEs? VAEs attain competitive sentence transformation performance compared to AEs in spite of the gap in reconstruction success. While the VAE underperforms the AE using linear transformation, the gap significantly narrows when using a cosine similarity-based pooling.

Transformer auto-encoders outperform existing methods in the literature. We compare against previous work in the literature that also transform sentences with latent vector edits. We use the publicly available implementations of DAAE and AAE, as well as Autobot (which we partially re-implement), and re-implement Steering Vectors, injecting the vector in layer 6 at all timesteps. We find that Steering Vectors is much better at the `neg_to_pos` than the `pos_to_neg` transformation on both sentiment datasets, which is consistent with what was reported by the authors.

Initializing from pre-trained LM checkpoints is highly beneficial. We find that although randomly initializing auto-encoders greatly improves reconstruction accuracy (Appendix **??**), we observe significant improvements in sentence transformation quality when initializing from

Tasks	Coefficient	Transformed Sentence
Negative-to-Positive Sentiment / Adding Qualifying Words	*Original*	*Possibly the worst editing I've ever seen in a movie!*
	$\alpha = 1.0$	Possibly the worst editing I've ever seen in a movie!
	$\alpha = 2.0$	Possibly the worst editing I've ever seen in a movie!
	$\alpha = 3.0$	Possibly the basically editing I've ever seen in a movie!
	$\alpha = 4.0$	Possibly basically the worst editing I've ever really seen in a movie!
	$\alpha = 5.0$	Possibly basically the best editing I've ever really seen in a movie!
Present-to-Simple Past / Prepended Phrase	*Original*	*The fanatic NAC fans immediately name the stadium the Rat Verlegh Stadion.*
	$\alpha = 1.0$	The fanatic NAC fans immediately, name the stadium the Rat Verlegh Stadion.
	$\alpha = 1.5$	Because of this fanatic, NAC fans immediately the stadium the Rat Verlegh Stadion.
	$\alpha = 2.0$	Because of this fanatic, NAC fans immediately named the stadium the Rat Verlegh Stadion.
	$\alpha = 2.5$	Because of this, fanatic NAC fans immediately named the stadium the Rat Verlegh Stadion.
	$\alpha = 3.0$	Because of this, this fanatic NAC immediately fans the stadium called the Rat Verlegh Stadion.

Table 4.6: Examples of composed transformations by applying the average transformation vector between two tasks.

pre-trained LM checkpoints. We conduct ablations over model initialization and tokenization by training randomly initialized models using split tokenization as well as a shared BERT tokenizer. We also evaluate a shared tokenizer model with partial (BERT-only) and complete (BERT and GPT-2) pre-trained initialization. We find that randomly initialized and shared tokenizer models are only somewhat comparable on additive sentence tasks, and are consistently outperformed by models that have both BERT and GPT-2 initialization.

Transformers are generally superior to LSTMs. Prior to the widespread adoption of Transformers, LSTM auto-encoders were studied mostly for their interpolation properties. Evaluations of analogy capabilities mostly centered around sentiment (and occasionally tense) transformations on Yelp. To put these results in a modern context, we evaluate two models that use 1-layer `LSTM` and 3-layer `BiLSTM` encoder-decoder architectures; the latent vector z is used to initialize the decoder state. We find that LSTM models perform modestly on the given tasks, but also that they are outperformed by a Transformer AE of comparable size (to the 3-layer BiLSTM.) We conclude that while LSTM AEs are capable of performing these analogy transformations, the Transformer architecture does confer benefits beyond pre-trained LM initialization.

Alternatives to the `memory_and_embeds` architecture perform comparably. We explore alternatives to the `memory_and_embeds` architecture of Li et al. [95], in particular on (a) inducing z using the encoder, and (b) passing z into the decoder. We evaluate generating z by pooling final layer activations (`avg_pool`), and passing z into the decoder by replacing the first token embedding with z. Our architecture ablations (Table 4.4) show that `avg_pool` and `sub_first_token` are comparable to `memory_and_embeds` on all tasks. This suggests that (1) pooling is a reasonable alternative to using the [CLS] token for z, and (2) the memory component is not strictly necessary.

Transformer VAEs scale better than AEs to increased model size. Increasing model size more than doubles the rate of reconstruction accuracy gain for Transformer VAEs (in terms of epochs required to reach the same accuracy.) Improvements in reconstruction accuracy also

Model	Spearman	Pearson
$\text{BERT}_{\text{BASE}}$ (average pooled)	**52.58**	**52.20**
GPT2-117M (average pooled)	22.68	13.78
GloVe (average pooled)	43.77	40.09
$\text{BERT}_{\text{BASE}}$ (`[CLS]` token)	26.43	24.98
GPT2-117M (last token)	9.71	1.12
Transformer AE	24.02	17.74
Transformer VAE	37.07	35.28

Table 4.7: The Spearman and Pearson correlations between the cosine similarity scores and the labels in STS-B for off-the-shelf LM embeddings and our Transformer auto-encoders.

translate to improved analogy accuracy. We note that even with a significant accuracy gap compared to the AE (75% vs 91%), the large VAE attains competitive analogy success on nearly all tasks.

Linear analogies are composable. We observe that many of our tasks can be composed by averaging their task vectors. We show some examples in Table 4.6. We perform some analysis of the transformation vectors and find evidence that similar tasks (in terms of vector similarity) also compose better. Compared to other models, we find that Transformer (V)AEs also compose better.

4.5.2 Semantic Textual Similarity

We evaluate the extent to which cosine similarity in the learned vector space corresponds to semantic similarity. We compare the vector representations produced by our auto-encoders with vector representations obtained by extracting (the start and end tokens as well as average pooling) the last-layer features from $\text{BERT}_{\text{BASE}}$ and GPT-2-117M, as well as GloVe embeddings [117] (Table 4.7). We find that the VAE features better represent semantic similarity compared to the AE features. We believe is because the VAE objective encourages a more isotropic representation. Despite being trained on an unrelated task, VAE attains better Spearman and Pearson correlation than all extractive embeddings other than average pooled BERT and GloVE, while the AE performs noticeably worse.

4.5.3 Biases: Occupation-Gender Associations

Past work has sought to characterize the gender associations that models learn from large text corpora. Word2Vec showed that occupation-gender associations exist in the linear structure of its vector space (such as *man*:*woman*::*king*:*queen*); subsequent work [17] found similar associations such as *man*:*programmer*::*woman*:*homemaker*. Recent work that investigates occupation-gender associations often uses coreference resolution tasks [100, 124] , i.e. is higher coreference score to “he” or “she” in the sentence “the doctor ran because ¡he/she¿ is late”? Motivated by these lines of inquiry, we study occupation-gender associations in the learned sentence embedding space. We select occupations from United Kingdom Census data[3] that we use to derive sentence templates that contain blanks for an occupation and gender pronoun. More details about this process can be found in Appendix **??**.

To evaluate occupation-based gender associations, we first use the aforementioned templates to generate masculine and feminine pronoun versions of sentences for each occupation, and then (interpolating between the two mean representations) compute the distance needed to induce the masculine-to-feminine (`m2f`) and feminine-to-masculine (`f2m`) transformations. We then compute the average interpolation distance for the `m2f` and `f2m` transformations across different templates. Finally, we test for monotonic trends in the interpolation distances ordered by their occupation's gender ratio in (a) the UK Census data the occupations were sourced from, or (b) the Wikipedia sentence corpus that the auto-encoder models were trained on. (For (b), we consider occupation-containing sentences that have exclusively male or female pronouns.) We use the non-parametric Mann-Kendall[4] trend test [79, 102] to test for monotonic trends on the `m2f` and `f2m` pronoun transformation distances, as well as their difference (`diff`). We find strong monotonic trends in all three trend lines (Figure **??**), and with respect to all orderings (Table **??**); although, we observe a stronger trend using UK Census occupation data. This shows that gendered representations of occupations have been learned by the auto-encoder models and are present in the latent space. Therefore, care should be taken to avoid perpetuating these biases when applying these models to tasks that have gender associations.

4.6 Conclusion

Our work revisited the idea of sentence auto-encoders in the modern Transformer era. To our knowledge, we have presented the first comprehensive study on latent space sentence transformations. We proposed novel techniques and analyses that highlight differences between the latent space induced by Transformer (V)AEs and by earlier LSTM models. These benefits make Transformer (V)AEs a promising candidate for many applications. However, we also uncovered some concerning behavior, such as the presence of occupation-gender biases, that should be taken into account if technology based on such models were to be employed.

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

[4] We use the `pyMannKendall` [66] package to perform our statistical tests.

Limitations

While our work focuses on exploring design decisions for integrating Transformer components into sentence auto-encoders, our reference architecture is largely based on an existing architecture from Li et al. [95]. Using this architecture requires integrating Transformer components that were trained separately and not designed to work together, e.g. BERT and GPT-2 have different tokenizers.

Our work does not attempt to modify the VAE setup, opting to use the standard Gaussian formulation and optimization procedures that are common in the literature. In addition, our methods for performing latent space manipulations are relatively straightforward extensions of previous work on analogy vectors from the Word2Vec era. While we show that a comparatively small amount of supervision (1-10 input pairs) is sufficient for our transformation scheme, it may be insufficient for more sophisticated transformation schemes.

0.31

(a) b

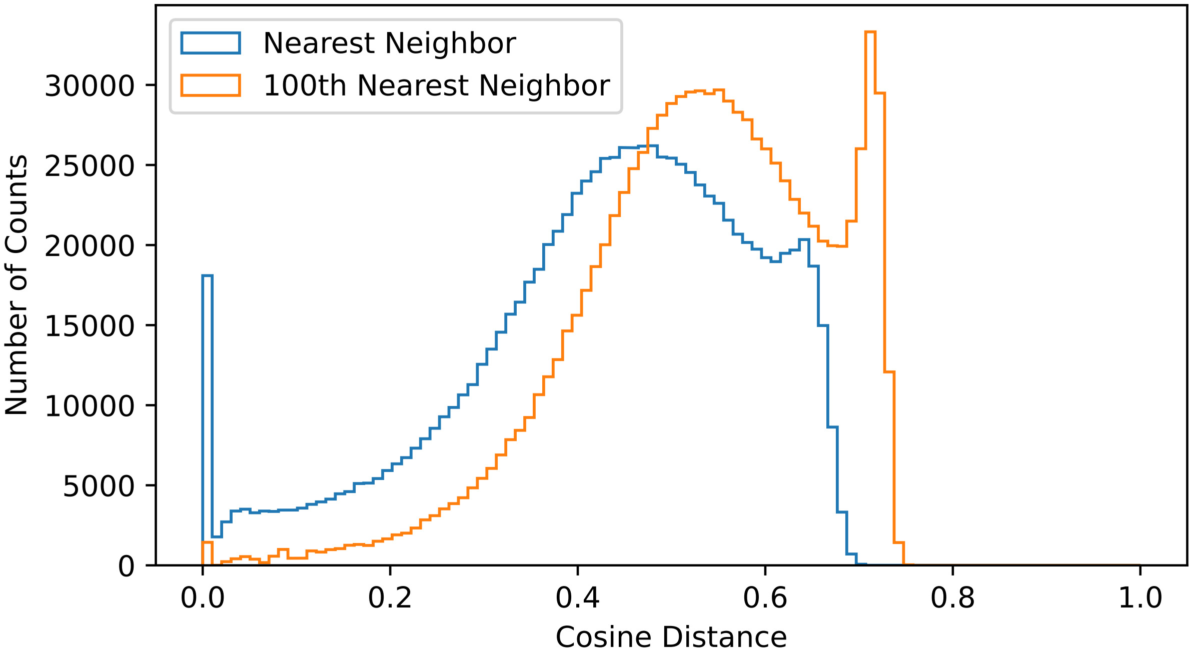


Figure 4.5: AE.

0.31

(a) b

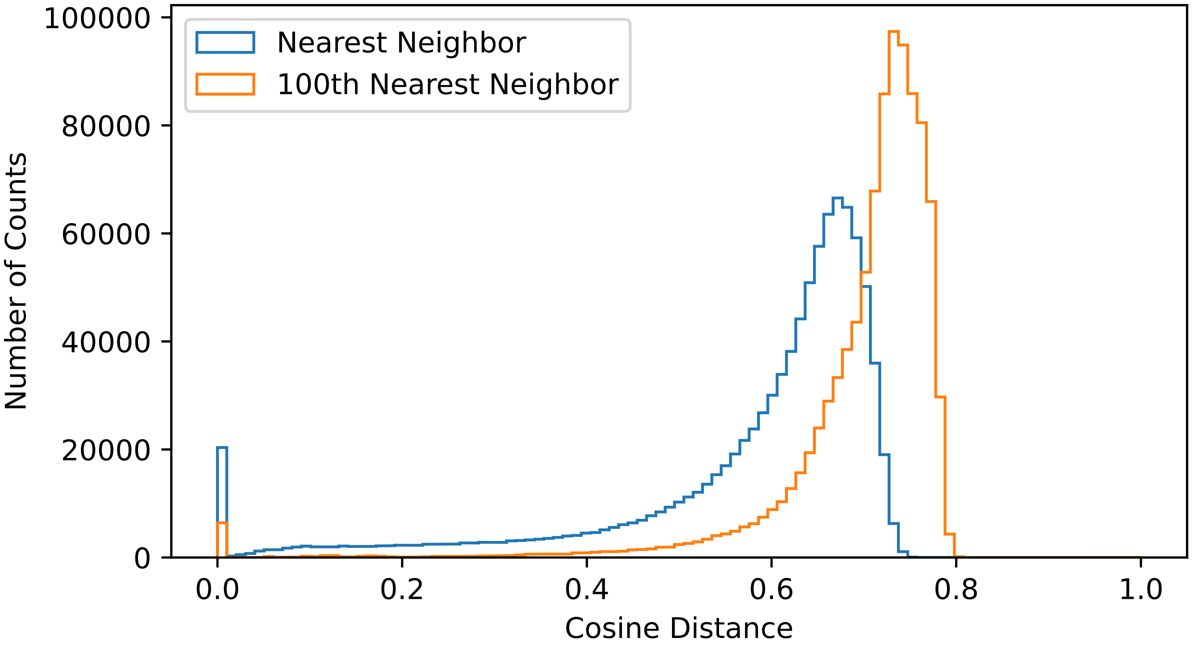
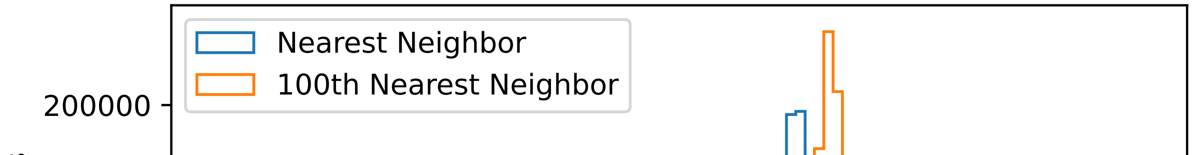


Figure 4.6: VAE mean.

0.31

(a) b



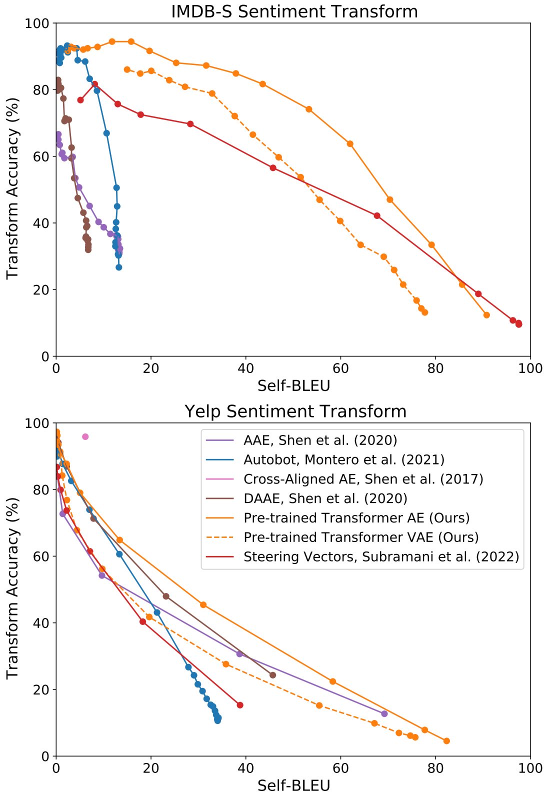


Figure 4.9: Sentiment transform results on the IMDB-S and Yelp datasets. We present the averaged result across the `neg_to_pos` and `pos_to_neg` tasks, and extrapolate the offset vector with coefficients $\lambda = \{1.0, 1.5, 2.0, 2.5, \ldots, 9.0, 9.5, 10.0\}$.

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`. 2.5

[2] Hervé Abdi. Holm's sequential bonferroni procedure. *Encyclopedia of research design*, 1 (8):1–8, 2010. 3.4

[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. 3.2

[4] 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`. 2.6.4, 2.6.4

[5] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to PMI-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399, 2016. doi: 10.1162/tacl_a_00106. URL `https://aclanthology.org/Q16-1028`. 4.4.2

[6] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002. 3.3.1

[7] Harriet E Baber. Adaptive preference. *Social Theory and Practice*, 33(1):105–126, 2007. 3.2

[8] 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. 2.1, 2.2

[9] Andrea Barraza-Urbina and Dorota Glowacka. Introduction to bandits in recommender

systems. In *Fourteenth ACM Conference on Recommender Systems*, pages 748–750, 2020. 3.1

[10] Soumya Basu, Rajat Sen, Sujay Sanghavi, and Sanjay Shakkottai. Blocking bandits. In *Advances in Neural Information Processing Systems*, pages 4785–4794, 2019. 3.2

[11] Manel Baucells and Rakesh K Sarin. Satiation in discounted utility. *Operations research*, 55(1):170–181, 2007. 3.2, 3.4

[12] Gary S Becker. *Accounting for tastes*. Harvard University Press, 1996. 3.4

[13] 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. 2.1, 2.3.2, 2.3.4, 2.6.4

[14] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. 1

[15] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007. 3.1

[16] Christopher M. Bishop. Mixture density networks. 1994. URL `http://publications.aston.ac.uk/id/eprint/373/`. 2, 2.4.3

[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. 4.5.3

[18] Tom Bosc and Pascal Vincent. Do sequence-to-sequence VAEs learn global features of sentences? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4296–4318, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.350. URL `https://aclanthology.org/2020.emnlp-main.350`. 4.4.2

[19] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015. 4.2, 4.2.1

[20] 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`. 4.1, 4.2

[21] GE Box. All models are wrong, but some are useful. *Robustness in Statistics*, 202(1979): 549, 1979. 3.6

[22] 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, 4.1

[23] Xingyu Cai, Jiaji Huang, Yuchen Bian, and Kenneth Church. Isotropy in the contextual

embedding space: Clusters and manifolds. In *International Conference on Learning Representations*, 2021. 4.4.2

[24] Leonardo Cella and Nicolò Cesa-Bianchi. Stochastic bandits with delay-dependent payoffs. In *International Conference on Artificial Intelligence and Statistics*, pages 1168–1177, 2020. 3.2

[25] 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`. 2.5

[26] 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. 2.2, 2.3.2

[27] 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. 3.1, 3.2, 3.6.2

[28] 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. 3.1, 3.2, 3.6.2

[29] 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. 3.1, 3.2, 3.6.2

[30] 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. 3.1, 3.2, 3.6.2

[31] 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

[32] Larry Christensen and Alisa Brooks. Changing food preference as a function of mood. *The journal of psychology*, 140(4):293–306, 2006. 3.2

[33] 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. 2.1

[34] 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 Sympo-*

sium 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. 2.1, 2.2, 2.4.2

[35] Eli P Cox III. The optimal number of response alternatives for a scale: A review. *Journal of marketing research*, 17(4):407–422, 1980. 3.1, 3.3.3

[36] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/842424a1d0595b76ec4fa03c46e8d755-Paper.pdf. 1

[37] 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. 2.2

[38] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 4.4.1

[39] Yuntian Deng, Anton Bakhtin, Myle Ott, Arthur Szlam, and Marc'Aurelio Ranzato. Residual energy-based models for text generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1l4SgHKDH. 4.1

[40] 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. 2.2

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423. 4.2

[42] Ronald M Epstein. Mindful practice. *Jama*, 282(9):833–839, 1999. 3.5.2

[43] Kawin Ethayarajh. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. 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 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1006. URL https://aclanthology.org/D19-1006. 4.4.2

[44] 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. 2.3.4

[45] Ronald Aylmer Fisher. Design of experiments. *Br Med J*, 1(3923):554–554, 1936. 3.4

[46] Brad Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004. 2.1

[47] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–28, 1994. 2.7

[48] Jeff Galak and Joseph P Redden. The properties and antecedents of hedonic decline. *Annual review of psychology*, 69:1–25, 2018. 3.1, 3.2, 3.4

[49] Jeff Galak, Jinwoo Kim, and Joseph P Redden. Identifying the temporal profiles of hedonic decline. *Organizational Behavior and Human Decision Processes*, 169:104128, 2022. 3.4

[50] 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. 2.1, 2.3.1

[51] 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. 2.2

[52] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003. 2.1

[53] GoComics. Gocomics. https://www.gocomics.com, 2021. 3.3.2

[54] 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. 3.1

[55] 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. 3.5.2

[56] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/

paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf. 1

[57] Paul Grossman. On measuring mindfulness in psychosomatic and psychological research. 2008. 3.5.2

[58] Philip M Groves and Richard F Thompson. Habituation: a dual-process theory. *Psychological review*, 77(5):419, 1970. 3.1

[59] Jiatao Gu, Kyunghyun Cho, and Victor O.K. Li. Trainable greedy decoding for neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1968–1978, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1210. URL https://aclanthology.org/D17-1210. 4.1

[60] Faruk Gul and Wolfgang Pesendorfer. The revealed preference theory of changing tastes. *The Review of Economic Studies*, 72(2):429–448, 2005. 3.2

[61] 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. 2.5

[62] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2.2

[63] 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. 3.2

[64] 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. 2.1

[65] John R Hurley and Raymond B Cattell. The procrustes program: Producing direct rotation to test a hypothesized factor structure. *Behavioral science*, 7(2):258, 1962. 4.4.1

[66] 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. 4

[67] Eugene Ie, Chih-wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. Recsim: A configurable simulation platform for recommender systems. *arXiv preprint arXiv:1909.04847*, 2019. 3.6.2

[68] 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. 3.1

[69] 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. 2.1, 2.3.2, 2.6.4

[70] 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. 3.1

[71] 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. 2.5

[72] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976. 4.4.1

[73] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018. 1

[74] 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. 2.1

[75] 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

[76] 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. 3.2

[77] Rafael-Michael Karampatsis and Charles Sutton. Maybe deep neural networks are the best choice for modeling source code. *arXiv preprint arXiv:1903.05734*, 2019. 2.3.3

[78] 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=SklgsONFvr. 4.4.3

[79] Maurice George Kendall. Rank correlation methods. 1948. 4.5.3

[80] 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. 2.1, 2.3.2

[81] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 4.1

[82] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016. 4.2, 4.2.1

[83] 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. 2.2

[84] 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

[85] Robert Kleinberg and Nicole Immorlica. Recharging bandits. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 309–319. IEEE, 2018. 3.2

[86] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 3.1

[87] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020. 3.1, 3.3.1, 3.4, 3.5.1

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

[89] 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. 3.2

[90] 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. 3.2

[91] 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. 3.2

[92] Liu Leqi, Fatma Kilinc-Karzan, Zachary C Lipton, and Alan L Montgomery. Rebounding bandits for modeling satiation effects. 2021. 3.2

[93] Nir Levine, Koby Crammer, and Shie Mannor. Rotting bandits. In *Advances in neural information processing systems*, pages 3074–3083, 2017. 3.2

[94] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703. 4.1

[95] 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. 4.1, 4.1, 4.2, 4.2.1, 4.3.1, 4.5.1, 4.6

[96] 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

[97] 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. 3.1

[98] 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. 2.2

[99] Fenrong Liu et al. *Changing for the better: Preference dynamics and agent diversity*. Institute for Logic, Language and Computation, 2007. 3.2

[100] 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. 4.5.3

[101] Florian Mai, Nikolaos Pappas, Ivan Montero, Noah A. Smith, and James Henderson. Plug and play autoencoders for conditional text generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6076–6092, Online, November 2020. Association for Computational Linguistics. doi:

10.18653/v1/2020.emnlp-main.491. URL https://aclanthology.org/2020.emnlp-main.491. 4.1, 4.2

[102] Henry B Mann. Nonparametric tests against trend. *Econometrica: Journal of the econometric society*, pages 245–259, 1945. 4.5.3

[103] 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. 3.1

[104] 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. 3.1, 3.2, 3.6.2

[105] 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. 3.1, 3.2, 3.6.2

[106] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 2.4.3, 4.2

[107] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1

[108] 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. 3.2

[109] 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), 4.1, 4.2, 4.4.3, 4.4

[110] Jiaqi Mu and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkuGJ3kCb. 4.4.2

[111] 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. 2.3.4

[112] OpenCensus. Opencensus. https://opencensus.io/, 2020. URL https://opencensus.io/. 2.1, 2.3.1

[113] 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`. 2.3.2

[114] 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`. 2.1, 2.2, 2.2, 2.3.4, 2.4.2

[115] Damian Pascual, Beni Egressy, Clara Meister, Ryan Cotterell, and Roger Wattenhofer. A plug-and-play method for controlled text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3973–3997, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.334. URL `https://aclanthology.org/2021.findings-emnlp.334`. 4.1

[116] 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. 3.5.1

[117] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL `https://aclanthology.org/D14-1162`. 4.5.2

[118] Drazen Prelec. Decreasing impatience: a criterion for non-stationary time preference and "hyperbolic" discounting. *Scandinavian Journal of Economics*, 106(3):511–532, 2004. 3.2

[119] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019. 2.2, 4.1, 4.2

[120] 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. 3.2

[121] 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. 4.1

[122] 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. 3.2

[123] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952. 3.1, 3.4

[124] Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. Gender 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`. 4.5.3

[125] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SkGuG2R5tm`. 4.4.2

[126] 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. 2.4.1

[127] 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. 3.2

[128] 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

[129] 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. 3.3.2, 3.4

[130] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. 4.4.1

[131] 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. 2.2, 2.3.3

[132] 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. 3.2

[133] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005. 3.1, 3.2, 3.6.2

[134] 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. 4.1, 4.4.3

[135] 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), 4.1, 4.2, 4.4.3, 4.4

[136] Wenxian Shi, Hao Zhou, Ning Miao, and Lei Li. Dispersed exponential family mixture vaes for interpretable text generation. In *International Conference on Machine Learning*, pages 8840–8851. PMLR, 2020. 4.2

[137] Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. On-the-fly adaptation of source code models using meta-learning. *arXiv preprint arXiv:2003.11768*, 2020. 2.3.3

[138] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop

distributed file system. In *MSST*, volume 10, pages 1–10, 2010. 2.1, 2.2

[139] 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/`. 2.1, 2.2

[140] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32, 2019. 1

[141] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019. 3.1, 3.4

[142] 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`. 2.2

[143] Michael Stonebraker and Greg Kemnitz. The postgres next generation database management system. *Communications of the ACM*, 34(10):78–92, 1991. 2.1

[144] 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, 4.4.3, 4.4

[145] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015. 3.1

[146] Engkarat Techapanurak, Masanori Suganuma, and Takayuki Okatani. Hyperparameter-free out-of-distribution detection using cosine similarity. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020. 4.4.1

[147] William T Tucker. The development of brand loyalty. *Journal of Marketing research*, 1 (3):32–35, 1964. 3.2

[148] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04): 376–380, 1991. 4.4.1

[149] 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. 2.2, 1, 4.1

[150] 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. 2.5

[151] Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1041–1049, 2017. 4.4.1

[152] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018. 4.4.1

[153] 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. 3.1, 3.2, 3.6.2

[154] 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. 4.4.3

[155] 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. 3.1

[156] 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. 2.1, 2.2

[157] 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. 2.7

[158] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *International conference on machine learning*, pages 3881–3890. PMLR, 2017. 4.2

[159] Lan Zhang, Wray Buntine, and Ehsan Shareghi. On the effect of isotropy on VAE representations of text. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 694–701, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.78. URL `https://aclanthology.org/2022.acl-short.78`. 4.4.2

[160] 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

[161] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176, 2018. 3.6.2

[162] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019. 1