LEARNED USER EMBEDDINGS FROM MODEL CONTEXT PROTOCOL TOOL INTERACTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model Context Protocol (MCP) is a mechanism that allows Small and Large Language Models to call external data sources in real-time and generate up-to-date and grounded responses for user prompts. MCP servers provide a discrete set of tools which are selected based on reasoning and semantic similarity to the user's prompt. Due to its discrete nature, use of MCP tools give a unique opportunity to understand user behavior, allowing a Language Model to build personalized experiences such as initial prompt suggestions and customized responses. In this paper, we show that dimension reduction techniques can be applied to build learned representations in the form of fixed size dense embeddings in latent space to represent MCP tool invocations. We apply three different dimensionality reduction techniques on normalized tool invocation counts: Principal Component Analysis (PCA), shared auto-encoder and truncated singular value decomposition. Using a formal approach to the problem we use an experimental data set and show how the learned user embeddings determine similarity between users providing a foundation for personalization related features.

Keywords: Machine learning, personalization, model context protocol, language models

1 INTRODUCTION

Large and Small Language Models (LM) have rapidly evolved into versatile systems capable of supporting a wide variety of user tasks, ranging from information retrieval to creative content generation and supporting decision making. Their general-purpose nature allows them to serve a broad set of use cases to diverse user populations, but this same generality inhibits the depth of personalization that can be made available to individual users. Personalization is defined as feature of an interactive software system to customize individual user experiences, with the primary goal of improving user satisfaction, task efficiency, and overall engagement Li & Karahanna (2007); Teevan et al. (2006). Although LMs are an interactive system, adapting LM behavior to specific user preferences in a principled and privacy-conscious way remains an open challenge. Currently, common personalization methods in language models rely on explicit feedback, static user profiles, or limited interaction history stored as memories, approaches that still fail to fully capture the richness and dynamics of user–LM interactions.

To provide personalized experiences in LMs we need to understand user intent, workflow preferences, and contextual needs. Unfortunately, most LMs have black-box opacity, and the lack of interpretability on how LMs infer the user's prompts make determination of intent and workflows difficult. The use of Retrieval Augmented Generation (RAG) techniques elevates this to some extent as each prompt response can be mapped to one of more discrete RAG calls, but RAG calls do not adhere to a structured and disciplined protocol, which makes determination of user attributes from them difficult.

With the advent of Model Context Protocol (MCP) we now have a formal structure and standardized interfaces to access data from external sources (similar to RAG), offering a promising avenue for personalization. A MCP server consists of one or more tools, each exposing a well-defined interface for accessing data or invoking external capabilities. Tool invocations can serve as implicit signals of user intent, workflow preferences, and contextual requirements.

By analyzing tool invocation patterns, it becomes possible to derive meaningful signals about how users interact with the system, what types of information they prioritize, and which external re-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

sources they most frequently depend upon. Unlike static preference data, tool invocation logs provide dynamic, context-sensitive insights that evolve organically with the user's ongoing activities.

Incorporating tool invocation data into personalization strategies opens up several research questions: How can such data be transformed into reliable models that aid in personalization ? More importantly, how can personalization be achieved while respecting privacy, transparency, and fairness? Addressing these questions requires not only technical innovations in user modeling and adaptive generation but also careful consideration of ethical implications in handling user data.

This paper contributes to this space by proposing a framework for personalizing user experiences in LMs by building user embeddings using MCP tool invocation patterns. These user embeddings can then be used to determine similarity between users, serving as a foundation to build personalized experiences. In doing so, our goal is to bridge the gap between general-purpose LM capabilities and individualized context-aware assistance. Importantly, the embeddings and the methods we have developed to generate them reduces exposure to sensitive data as no Personally Identifiable Information (PII) is stored or used to generate then.

In the next sub-section, we give more details on MCP followed by a description of user embeddings and how they can power personalization features. Section 2 shows the related work on LM personalization, in section 3 we mention the concepts and assumption in our approach followed by formal definitions. Section 4 provides implementation details on the dimensionality reduction algorithms, followed by experimental results (Section 5). We conclude with a summary (Section 6) and opportunities for future work (Appendix).

1.1 Model Context Protocol

The Model Context Protocol (MCP) MCP (2024) is an open-source framework that enables language models — both small and large to interface with external data sources in real-time. Rather than relying solely on static pre-trained knowledge, MCP introduces a server-driven architecture where external "tools" can be registered and invoked dynamically by language models. These tools represent discrete computational or data-access functions, allowing the model to ground responses with authoritative and latest data. There are no protocol specified limits on the number of tools each MCP server can have, though at least one tool is required.

A core principle of MCP is it's semantic and reasoning-driven tool invocation: the model maps user prompts to relevant tools based not only on keyword matching but also on semantic similarity and task alignment. This design ensures modularity, enabling MCP developers to add or update tools independently of the model's training process, thereby extending the model's effective capabilities without any expensive retraining or fine-tuning.

MCP Servers can be local using *stdio* protocol, or are accessible over the network with the *StreamableHTTP*. The metadata for tools available in a server are generally fetched at run time (using a MCP 'tools/list' call) by the LM. Each MCP tool is described by a well-defined schema that includes metadata essential for discovery, invocation, and safe execution. Some of the important metadata fields are:

Metadata	**Description**	**Example**
Name	A unique name that identifies the tool	get_weather
Title	User friendly tool name	Weather Information Provider
Description	Detailed description of the tool	Get current weather information for a location
InputSchema	Structured JSON schema of tool inputs	{ "type": "object", "properties": { "location": { "type": "string", "description": "City name or zip code" } }, "required": ["location"] }

This metadata-driven design allows MCP servers to expose a diverse set of tools to models in a self-describing manner, much like an API registry but optimized for model reasoning. In our example, the 'get_weather' tool carries metadata specifying its ability to take parameters such as *location: string*, with a return value describing temperature and general weather conditions. Such metadata not only guides the model during inference, but also enforces guardrails by making input/output expectations explicit, reducing the risk of malformed calls or ambiguous responses. In practice, MCP

transforms language models from closed-text generators into extensible reasoning systems capable of orchestrating multiple external resources through formally defined, semantically discoverable interfaces.

The *description* and *InputSchema* fields have the most important role for the tool being selected. Once a user's intent is determined from the prompt, LMs perform a semantic similarity search between the user's intent and the description field of the MCP tools. Based on the chosen similarity measure and the inputs available, a tool is selected to provide the data required to fulfill the user prompt. As an example, if the user's prompt to the LM is:

"What is the temperature in San Francisco right now?"

The LM determines the intent as information on temperature of the given location of "San Francisco". The LM then analyzes all the MCP tools available, analyzes the *Description* and *InputSchema* fields, and ranks those where the description matches 'providing temperature' and which take a single input.

In addition, tool metadata can optionally be enriched with annotation hints. The four current annotations are *readOnlyHint*, *destructiveHint*, *idempotentHint*, and *openWorldHint*. These hints tell the LM what kind of action or state change the tool can perform. The MCP protocol is undergoing continuous enhancements, and a deep dive on MCP is outside the purview of this paper; readers are directed to the formal documentation MCP (2024).

1.2 User Embeddings and Personalization

Embeddings are vector representations of discrete entities (such as words, sentences, or user behaviors) that capture semantic or relational similarities in a continuous space. They allow models to understand complex relationships between entities efficiently by placing similar entities closer together in the multi-dimensional space.

If embeddings of all users in a LM system are determined, it enables personalization and pattern discovery. Classical statistical and machine-learning models can be built to identify similar users, recommend relevant content, cluster user groups, and detect anomalies or behavioral trends, all while maintaining scalability and computational efficiency and importantly, reducing exposure of sensitive user data.

To generate embeddings for users, some measurable data that can be transformed to numeric representations has to be available. Previously in LMs with just user prompts, the only available data was the number of times user logging in, the number of tokens being used etc., but these do not provide the granularity to understand user's behavior. With MCP this is no longer the case as each tool call serves a specific purpose and the number of tool calls made by each user can be accurately measured. A MCP tool does not represent Personally Identifiable Information (PII) of any user so using only it's invocation count also reduces exposure of sensitive data. Note that the inputs provided to the tool may contain PII, but the inputs are not considered when we generate the embeddings.

2 Related Work

Zhang et al. (2025) and Tseng et al. (2024) provide a comprehensive survey of current state-of-the-techniques in LM personalization. To give better context of our work, we have highlighted relevant work in sections dealing with general Language Model personalization and those that generate user embeddings which are applied to personalization.

2.1 Language Model Personalization

The current focus of LM personalization has been fine-tuning user specific models or adding user context during the retrieval process.OPPU Tan et al. (2024b) adopt the LoRA Hu et al. (2022) method to tune a Llama model Touvron et al. (2023) for each user, and it's efficiency is further improved in Tan et al. (2024a). Zhang et al. (2024) apply PEFT in a memory injected approach. Hydra Zhuang et al. (2024) is a learning-based framework that captures user-specific behavior using a re-ranker.

Salemi et al. (2024) explore different retrieval models and their efficiency in personalization. Richardson et al. (2023) present a summary-augmented approach, and augmenting retrieval with

user interest journey is presented in Christakopoulou et al. (2023), while Liu et al. (2023) present RETA-LLM retrieval augmented toolkit that can be used for personalization.

Further, Persona based adaption by LLMs have been shown in Cho et al. (2022) who use a variational auto-encoder to learn from dialogue history, and Wu et al. (2024) show the benefits of user profiles. None of these methods use interaction based user-embeddings which can identify similar users in a computationally efficient manner.

2.2 User Embeddings for Language Models

Ning et al. (2024) develop a novel approach of using user's interaction history into dense embeddings which are then integrated directly into the LLM using cross-attention and treated like another modality. They use the temporal order of interactions but the interactions are captured with items which need independent features of their own (e.g. for movies the features will be the title, genre, rating etc.). In our case, MCP tools need no predefined metadata because both proposed methods learn directly from the user-tool interaction matrix. Liu et al. (2024) introduce Persona-Plug that generates a personal embedding per user. These are generated by encoding past natural language interactions (reviews, tweets etc.) into a dense vector, then a different weightage is assigned to each behavior using an aggregator function. Experiments have been provided on the LaMP benchmark Salemi et al. (2024). Doddapaneni et al. (2024) compress the user history of movie selections into soft-prompt embeddings using text-to-text transformers.

Our approach of using discrete tool calls is much simpler and requires less compute. Another concern with most of the previous approaches is that embeddings are generated based on user's natural language interactions which can violate privacy laws in certain countries and our approach of using just the tool call count preserves privacy.

2.3 Collaborative Filtering and Raw Distance Measures

Collaborative Filtering learns user preferences from patterns in a user-item interaction matrix rather than from explicit item metadata Koren et al. (2009). Our user-tool matrix has the same structure: users correspond to consumers, MCP tools correspond to items, and normalized invocation counts provide implicit feedback. The resulting latent user factors can therefore identify users with similar behavior, although our objective is to support broader language-model personalization rather than recommend tools directly.

Principal Component Analysis (PCA) Maćkiewicz & Ratajczak (1993) is another matrix-factorization method for dimensionality reduction. Because PCA is sensitive to feature scaling, tools with large invocation counts can dominate its components. Distances can also be computed directly from tool-count vectors, although these vectors have one coordinate per tool and are therefore neither compact nor dense. We use standardized PCA and raw tool-count distances as baselines, and truncated SVD as one of the proposed embedding methods.

3 Basic Concepts and Formal Definitions

We introduce the important concepts, and notations used to denote these concepts in the rest of the paper. In addition, we discuss assumptions made for our study.

3.1 Basic Concepts

A *prompt* is an input in natural language given to a LM that guides it to generate a specific response or output. A *session* is a continuous interaction between a user and a language model represented by multiple prompts and their corresponding responses. Prompts given in a session follow a temporal order where the *context* of a downstream prompt includes the prior prompts and their responses, allowing the session to maintain coherence and contextual memory. To generate the response of the prompt a LM can call one or more MCP tools, hence a session can also be represented as a temporal sequence of tool calls.

The inputs required for a tool are extracted by the LM based on it's available context, and the availability of the input field in context is a factor in selecting tools. *Users* are humans that interact with the LM, with each user having a unique identity and no assumed upper bound on their cardinality.

3.2 ASSUMPTIONS

In a LM not all prompts will map to a tool call in which case we use a null tool to represent the mapping to the prompt. Like unique ids for each user, every session has a unique id with a one-to-one mapping with the user who created the session. Each tool call is also associated with a session allowing each tool call to be traced to a user. We assume that this information on sessions, tools and user is stored in a separate data store maintained by the LM and available at the time of building embeddings.

Most MCP servers give access to only authenticated users who are allowed to run the tools. We do not differentiate between MCP servers that require authentication and those that don't, and it is assumed that any authentication related workflows are handled internally by the LM.

In machine learning systems, a cold start problem is when the system lacks enough data to make recommendations. For user embeddings, cold start will be an issue and unless a threshold is minimum samples/user is defined all new users will get similar embeddings. This will inhibit personalized experiences for new users, in future work we will explore approaches to mitigate this.

3.3 FORMAL DEFINITIONS

- Let LM represent a small or large language model capable of executing MCP tools.
- Let M^{all} = {M_{null}, M_1, M_2, M_3 ... M_m} be a set of MCP servers, that can be called from LM.$|M^{all}| > 0$. M_{null} is a special case where LM does not use any MCP Server when responding to a user's prompt.
- T_i = {τ_1^i,τ_2^i,..., $\tau_{|T_i|}^i$} be the set of tools present in MCP server $M_i \in M^{all} \mid |T_i| > 0, T_i \subset M_i$. For M_{null} there is a single tool τ^{null}.
- T^{all} = $\bigcup_{i=1}^{|M^{all}|} T_i$, be the set of all tools present in LM.
- $I_i^j = \{in_1, in_2, \ldots, in_n\}$ be the inputs required for tool $\tau_i^j, |I_i^j| \geq 0$.
- U^{all} = {u_1,u_2,..., u_s} be the set of users in LM, $|U^{all}| > 0$
- S_i^n = {$\tau_1^m, \tau_2^m, ..., \tau_t^{'}, \tau_{t+1}^{'}, \tau_t^m$} are the list of tools used in a session, S_i^n denotes the n^{th} session for user u_i. Each session is a chain of prompts where each prompt invokes any of the available MCP tools in temporal order $\mid \tau_t^m \in T_m, T_m \subset M_i, 0 < x \leq |M^{all}|$.
- For any two $\tau_t^m, \tau_{t+n}^m \mid n > 0$ present in S_i^n, τ_t^m is called before τ_{t+n}^m.
- S_i is the set of all sessions in LM that were created by user u_i: $S_i^n \in S_i, 0 < n \leq |S_i|$.
- S^{all} denotes all the sessions in LM. Hence, $S_i \in S^{all} \mid 0 < i < |U^{all}|$.
- $E_i = (e_1^i, e_2^i, ..., e_d^i) \in \mathbb{R}^d$ is the embedding vector for user $u_i \in |U^{all}|$. d is the dimensionality of the vector embeddings.

4 ALGORITHMS

We first normalize the data to prevent a few MCP calls to dominate the embedding representations. Since PCA is already well-known, we focus on other methods which use autoencoder (Algorithm 2) and truncated singular value decomposition (Algorithm 3), both of which use the unit normalized $\hat{C}_1(u_i)_k$ values produced by Algorithm 1. Pseudo-code for algorithms is in appendix.

To illustrate embedding generation, let us first build a matrix where each row of $\hat{C}_1(u_i)_k$ is placed under the column allocated to each tool, with the Matrix being of size $|U^{all}|$ x $|T^{all}|$. The following is a representation of this matrix (referred to as $MAT^{u,\tau}$),

$$MAT^{u,\tau} = \begin{array}{c|cccc} & \boldsymbol{\tau_1^m} & \boldsymbol{\tau_2^m} & ... & \boldsymbol{\tau_{|T|}^m} \\ \hline \boldsymbol{u_1} & \hat{C}_1(1)_1 & \hat{C}_1(1)_2 & ... & \hat{C}_1(1)_{|T|} \\ \boldsymbol{u_2} & \hat{C}_1(2)_1 & \hat{C}_1(2)_2 & ... & \hat{C}_1(2)_{|T|} \\ \boldsymbol{u_3} & \hat{C}_1(3)_1 & \hat{C}_1(3)_2 & ... & \hat{C}_1(3)_{|T|} \\ ... & ... & ... & ... & ... \\ \boldsymbol{u_{|U|-1}} & \hat{C}_1(|U|-1)_1 & \hat{C}_1(|U|-1)_2 & ... & \hat{C}_1(|U|-1)_{|T|} \\ \boldsymbol{u_{|U|}} & \hat{C}_1(|U|)_1 & \hat{C}_1(|U|)_2 & ... & \hat{C}_1(|U|)_{|T|} \end{array}$$

here, $u_i \in U^{all}$, $\tau_i^m \in T^{all}$.

The embedding dimensions for each user can also be represented as a matrix, while we refer to as MAT^E.

$$MAT^E = \begin{array}{c|cccc} \boldsymbol{u_1} & e_1^1 & e_2^1 & ... & e_d^1 \\ \boldsymbol{u_2} & e_1^2 & e_2^2 & ... & e_d^2 \\ \boldsymbol{u_3} & e_1^3 & e_2^3 & ... & e_d^3 \\ ... & ... & ... & ... & ... \\ \boldsymbol{u_{|U|-1}} & e_1^{|U|-1} & e_2^{|U|-1} & ... & e_d^{|U|-1} \\ u_{|U|} & e_1^{|U|} & e_2^{|U|} & ... & e_d^{|U|} \end{array}$$

Our goal is to learn a mapping from $MAT^{u,\tau}$ to a compact shared representation MAT^E. Unlike an independently fitted vector for each user, a shared mapping places all users in the same coordinate system, so distances between rows of MAT^E are meaningful. We consider a nonlinear reconstruction model in Algorithm 2 and a linear low-rank model in Algorithm 3.

4.1 Shared Autoencoder

Let $X = MAT^{u,\tau} \in \mathbb{R}^{|U|\times|T|}$. Algorithm 2 uses one autoencoder for the complete user population. Its encoder maps every user-tool vector to d dimensions:

$$H = \mathrm{ReLU}(XW_1^e + b_1^e), \tag{1}$$

$$MAT^E = HW_2^e + b_2^e. \tag{2}$$

The decoder maps the embedding back to tool space:

$$\hat{X} = \mathrm{sigmoid}\left(\mathrm{ReLU}(MAT^E W_1^d + b_1^d)W_2^d + b_2^d\right). \tag{3}$$

The hidden width is a hyper-parameter and set to $h = \min(256, \max(2d, \min(|T|, 64)))$.

This provides a minimum of 64 hidden units based on input size, while keeping the hidden layer at least twice the embedding dimension, and the 256 caps memory and computation for large embeddings, an important consideration in implementation.

Thus, $W_1^e \in \mathbb{R}^{|T|\times h}$ and $W_2^e \in \mathbb{R}^{h\times d}$; the decoder reverses these dimensions.

Most entries of X are zero, so an unweighted reconstruction loss can favor the inactive tools. We instead use

$$\mathcal{L}_{AE} = \frac{1}{|U||T|}\sum_{i=1}^{|U|}\sum_{j=1}^{|T|} w_{ij}(\hat{X}_{ij} - X_{ij})^2, \quad w_{ij} = \begin{cases} 10, & X_{ij} > 10^{-4}, \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

This weighted loss allows an active tool's error to contribute ten times as much as an inactive tool's error.

For our experiments, the network is trained jointly over shuffled mini-batches of 256 users with Adam, a learning rate of 0.02, and random seed 1. Training stops when the epoch loss falls below 5×10^{-5} or after defined epochs (set to 5000 in our experiments). The encoder output after training is MAT^E, and the per-user weighted reconstruction MSE is retained for evaluation.

4.2 Truncated-SVD Matrix Factorization

Algorithm 3 computes a rank-d approximation of the same user-tool matrix:

$$X \approx U_d \Sigma_d V_d^T. \tag{5}$$

The user embedding matrix is

$$MAT^E = U_d\Sigma_d \in \mathbb{R}^{|U|\times d}, \tag{6}$$

and $V_d^T \in \mathbb{R}^{d\times|T|}$ contains the learned tool factors. The reconstructed interaction matrix is $\hat{X} = MAT^E V_d^T$. All user vectors therefore share the same latent axes. We use randomized truncated SVD with seed 1 and seven power iterations; d must not exceed $\min(|U|, |T|)$. For user u_i, reconstruction error is

$$loss_i = \frac{1}{|T|}\sum_{j=1}^{|T|}(\hat{X}_{ij} - X_{ij})^2. \tag{7}$$

4.3 Finding Similar Users

Once MAT^E is known, similarity between users is determined using methods that measure distances between vectors and we have used both cosine and euclidean distances allowing comparisons between the two.

Given MAT^E the cosine distance (cos_dist) and euclidean distance (euc_dist) between two users u_i and u_j is given by the following:

$$cos_dist(u_i, u_j) = 1 - \frac{(MAT^E_{i,:}).(MAT^E_{j,:})}{\|MAT^E_{i,:}\|\|MAT^E_{j,:}\|},$$

$\text{cos_dist} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0, 2]$.

and,

$$\text{euc_dist}(u_i, u_j) = \sqrt{\sum_{k=1}^{d}(MAT^E_{i,k} - MAT^E_{j,k})^2},$$

$\text{euc_dist} : \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0, \infty)$

5 Experimental Results

Currently commercially available MCP servers have propriety data which we cannot disclose in this paper, and existing personalization benchmarks like LaMP Salemi et al. (2024) do not provide MCP tool invocation data. Hence, for evaluation of our methods we have generated new synthetic data in a SQLLite database which has also been shared publicly to allow wider use for similar studies. Data has been generated for 10,000 users and 100 MCP servers with the number of tools in each MCP server varying between 1 and 50. The number of sessions per user is selected from a normal distribution with mean of 100 and Standard Deviation (SD) of 60. The length of a session is selected from another normal distribution of mean of 20 and SD of 10. These values are loosely based on tool availability in commercially available MCP servers. The code to generate this synthetic data and for all the 3 algorithms is available in cod, while the SQLite database with the generated synthetic data is available at dat.

Within each session, the first tool to be called is selected using two different random functions. The first function selects the MCP server M_i and a tool τ_{i1} within it which is called at the start of the session. M_{null} can be selected just like any other MCP server. Tools at depth $d > 0$, are selected by another random function which is similar except it takes the previous tool call into consideration. The probability of selecting a tool from the same MCP server used in depth $d - 1$ is set to 0.33, to account for the practical scenario where the next prompt is more likely to invoke a tool similar in nature from the same MCP server.

Results have been computed for embedding dimensions of 8 and 32.

5.1 Canary Users

For a sample of users we have overwritten the randomly generated data so the sample has known attributes that help us determine the effectiveness of user embeddings. Data of 10% of randomly selected user's are modified to have very similar session and tool invocations. We classify these samples as canary users, which are further sub-divided in two types.

5.1.1 Canary Users 1

Composing 5% of the users, all users in this group have same number of sessions and each session has same tool calls. It is expected that embeddings for these users should be exact, making their cosine and euclidean distance zero.

5.1.2 Canary Users 2

Composing the other 5% of the users, all users in this group have the same number of sessions but the depth of randomly selected sessions has been decreased by 1. The cosine and euclidean distances in this group are expected to be close but not zero.

5.2 Distance and Clustering

In Figure A3, the main characteristics of the synthetic data are shown. These are the number of tools in each MCP server and the distribution of sessions and tool calls. From Figure A3(c), we observe a skew with some tools used more frequently than others. This reflects practical scenarios where tools for common functions (e.g. summarization) will be used more often than others.

Results for embedding dimensions of 8 and 32 are shown separately. For visualization only, we project each learned embedding matrix to two principal components; these projections are shown in Figures 1 and A5. Additional distance and clustering plots are provided in the Appendix. The mean reconstruction losses for Algorithm 2 were 2.6345×10^{-5} at both dimensions. Algorithm 3 obtained lower losses of 1.9118×10^{-6} at dimension 8 and 1.2919×10^{-6} at dimension 32.

Canary 1 users collapse to a single point for both algorithms in Figures 1(b), 1(e), A5(b), and A5(e). Relative to the projections of all users, Canary 2 embeddings are also tightly grouped. The distance measurements below confirm both observations rather than relying on the two-dimensional projections alone.

5.2.1 Clustering and topK users

In personalization, knowing the distinct clusters of users helps determine distinct attributes like persona, corporate roles etc. Using the elbow in the Within-Cluster Sum of Squares (WCSS) curve, we determined an operating cluster count for embedding sizes 8 and 32. This is the point of diminishing returns in WCSS, rather than the trivially lowest WCSS, which occurs at the largest tested value of k.

Retrieving the nearest users is useful for cohort-based personalization. Table A1 reports the top-5 nearest neighbors of the non-canary user u_1. The neighbors are sorted by increasing distance and are computed directly in each method's output space. Cosine and Euclidean distance need not produce the same ordering because the latter is also sensitive to vector magnitude.

5.3 Baseline Comparisons

PCA and raw tool-count vectors provide simple reference points for the learned representations. We use them to isolate the effect of dimensionality reduction rather than to compare model complexity. The PCA baseline standardizes the original tool-count features before projection; the raw baseline retains all 2,465 tool-count dimensions.

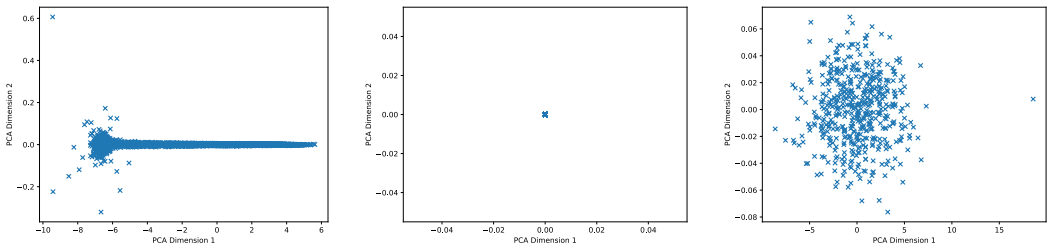
5.3.1 Distance Measures

For each canary group and for the full population, we computed every unordered user pair and report the mean (μ) and population standard deviation (σ). Before measuring distance, Algorithm 2 and 3 coordinates were min-max scaled per dimension and all representations were L2 normalized per user. The common normalization bounds the Euclidean scale while cosine distance remains determined by vector direction; these results are shown in Table A2.

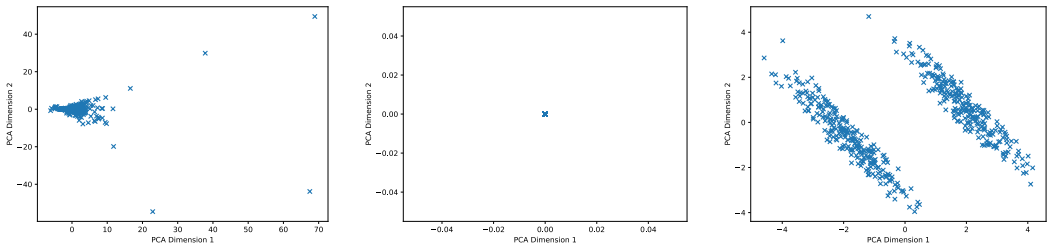
5.3.2 Clustering

Table A3 gives the elbow-selected cluster count and its silhouette score. A higher silhouette score normally indicates better separation, but it must be read together with cluster sizes.

Algorithm 2 consistently selects four clusters and has a silhouette score near 0.56, slightly above PCA and substantially above the raw-count baseline. Algorithm 3 produces much smaller normalized distances and the lowest reconstruction loss, but its clustering result is degenerate: at dimension 8, 9,992 users occupy one cluster and the remaining eight users are split across seven clusters; at dimension 32, one cluster contains 9,999 users and the other contains a single user. Its high silhouette scores therefore reflect outlier isolation rather than useful population segmentation. Figure 2 shows that this behavior persists across the tested cluster counts. This distinction is important: truncated SVD is effective for reconstruction and nearest-neighbor retrieval, while the nonlinear autoencoder provides the more balanced clustering geometry in this experiment.

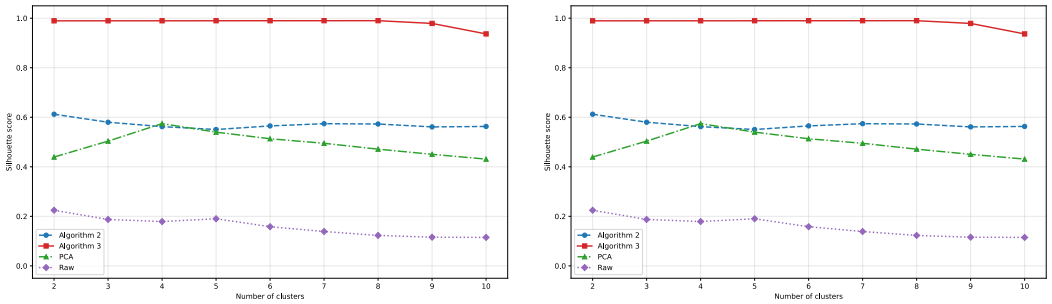


(a) Algorithm 2: All Users (b) Algorithm 2: Canary 1 Users (c) Algorithm 2: Canary 2 Users



(d) Algorithm 3: All users (e) Algorithm 3: Canary 1 Users (f) Algorithm 3: Canary 2 Users

Figure 1: Embeddings size 8: PCA for Algorithms 2 and 3



(a) Embedding dimension 8 (b) Embedding dimension 32

Figure 2: Silhouette scores for Algorithms 2 and 3 over cluster counts 2–10, computed from all 10,000 users. Algorithm 3's high values should be interpreted with its highly imbalanced cluster sizes.

6 Conclusions

In this paper, we presented methods for generating low-dimensional user embeddings from MCP interactions. Algorithm 1 prepares and normalizes tool counts, Algorithm 2 learns a shared nonlinear autoencoder, and Algorithm 3 obtains user factors through truncated SVD. Experiments on synthetic data show that the resulting spaces support similarity search, although their clustering properties differ.

Users with identical tool histories (Canary 1) receive effectively identical embeddings, while the perturbed Canary 2 users remain close under both distance measures. Increasing the embedding dimension from 8 to 32 did not improve Algorithm 2's reconstruction loss and only modestly reduced its all-user distances. Truncated SVD reconstructed the interaction matrix more accurately, but its apparent clustering strength arose from isolated outliers. These results favor the autoencoder when balanced cohort discovery is required and SVD when compact linear reconstruction or neighbor retrieval is the primary objective. Further work is needed to test these conclusions on non-synthetic interaction logs and to select embedding dimension as the user and tool populations grow.

Additionally the synthetic data we generated has been shared allowing other researchers to utilize the data in their work.

REFERENCES

Experiments code repository. https://anonymous.4open.science/r/Research-C5AE/UserEmbeddings_MCP/.

Synthetic data repository. https://limewire.com/d/fsqxm#XiR4NXwFhl.

Model context protocol. https://modelcontextprotocol.io/, 2024. (Accessed: Sept. 22, 2025).

Itsugun Cho, Dongyang Wang, Ryota Takahashi, and Hiroaki Saito. A personalized dialogue generator with implicit user persona detection, 2022. URL https://arxiv.org/abs/2204.07372.

Konstantina Christakopoulou, Alberto Lalama, Cj Adams, Iris Qu, Yifat Amir, Samer Chucri, Pierce Vollucci, Fabio Soldo, Dina Bseiso, Sarah Scodel, Lucas Dixon, Ed H. Chi, and Minmin Chen. Large language models for user interest journeys, 2023. URL https://arxiv.org/abs/2305.15498.

Sumanth Doddapaneni, Krishna Sayana, Ambarish Jash, Sukhdeep Sodhi, and Dima Kuzmin. User embedding model for personalized language prompting, 2024. URL https://arxiv.org/abs/2401.04858.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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

Shibo Li and Elena Karahanna. Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *Journal of Management Information Systems*, 23(3): 45–70, 2007.

Jiongnan Liu, Jiajie Jin, Zihan Wang, Jiehan Cheng, Zhicheng Dou, and Ji-Rong Wen. Reta-llm: A retrieval-augmented large language model toolkit, 2023. URL https://arxiv.org/abs/2306.05212.

Jiongnan Liu, Yutao Zhu, Shuting Wang, Xiaochi Wei, Erxue Min, Yu Lu, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. Llms + persona-plug = personalized llms, 2024. URL https://arxiv.org/abs/2409.11901.

Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993. ISSN 0098-3004.

Lin Ning, Luyang Liu, Jiaxing Wu, Neo Wu, Devora Berlowitz, Sushant Prakash, Bradley Green, Shawn O'Banion, and Jun Xie. User-llm: Efficient llm contextualization with user embeddings, 2024. URL https://arxiv.org/abs/2402.13598.

Chris Richardson, Yao Zhang, Kellen Gillespie, Sudipta Kar, Arshdeep Singh, Zeynab Raeesy, Omar Zia Khan, and Abhinav Sethy. Integrating summarization and retrieval for enhanced personalization via large language models, 2023. URL https://arxiv.org/abs/2310.20081.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large language models meet personalization, 2024. URL https://arxiv.org/abs/2304.11406.

Zhaoxuan Tan, Zheyuan Liu, and Meng Jiang. Personalized pieces: Efficient personalized large language models through collaborative efforts. *CoRR*, abs/2406.10471, 2024a. doi: 10.48550/ARXIV.2406.10471. URL https://doi.org/10.48550/arXiv.2406.10471.

Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing large language models via personalized parameter-efficient fine-tuning. *CoRR*, abs/2402.04401, 2024b. doi: 10.48550/ARXIV.2402.04401. URL https://doi.org/10.48550/arXiv.2402.04401.

Jaime Teevan, Susan T. Dumais, and Daniel J. Liebling. A study on the effects of personalization and task information on implicit feedback performance. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 449–458. ACM, 2006.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi: 10.48550/ARXIV.2302.13971. URL https://doi.org/10.48550/arXiv.2302.13971.

Yu-Min Tseng, Yu-Chao Huang, Teng-Yun Hsiao, Wei-Lin Chen, Chao-Wei Huang, Yu Meng, and Yun-Nung Chen. Two tales of persona in llms: A survey of role-playing and personalization, 2024. URL https://arxiv.org/abs/2406.01171.

Bin Wu, Zhengyan Shi, Hossein A. Rahmani, Varsha Ramineni, and Emine Yilmaz. Understanding the role of user profile in the personalization of large language models, 2024. URL https://arxiv.org/abs/2406.17803.

Kai Zhang, Lizhi Qing, Yangyang Kang, and Xiaozhong Liu. Personalized llm response generation with parameterized memory injection. *arXiv preprint arXiv:2404.03565*, 2024.

Zhehao Zhang, Ryan A. Rossi, Branislav Kveton, Yijia Shao, Diyi Yang, Hamed Zamani, Franck Dernoncourt, Joe Barrow, Tong Yu, Sungchul Kim, Ruiyi Zhang, Jiuxiang Gu, Tyler Derr, Hongjie Chen, Junda Wu, Xiang Chen, Zichao Wang, Subrata Mitra, Nedim Lipka, Nesreen Ahmed, and Yu Wang. Personalization of large language models: A survey, 2025. URL https://arxiv.org/abs/2411.00027.

Yuchen Zhuang, Haotian Sun, Yue Yu, Rushi Qiang, Qifan Wang, Chao Zhang, and Bo Dai. Hydra: Model factorization framework for black-box llm personalization, 2024. URL https://arxiv.org/abs/2406.02888.

APPENDIX

FUTURE WORK

This work sets up a foundation for further work on embedding generation based on MCP tool calls for the purpose of LLM personalization and usage pattern discovery as a downstream activity. Future work will focus on the following areas:

- Apart from absolute tool count, considering the temporal ordering of tool calls will lead to more representative embeddings.
- In this work, we limited our scope to only two embedding sizes. A more comprehensive study is needed where the change in metrics is measured for an incremental change in embedding size to find an optimum number.
- Like other machine learning based systems, cold start remains an issue in generating embeddings for new users. Future work will explore methods to use approximate embeddings till sufficient tool usage data is organically available.

AI DISCLOSURE

In this work, we have used generative AI tools for finding related work and understanding their technical contributions. Code for experimentation is also generated with assistance from LLM and human-reviewed for correctness by all authors. We have not used generative AI for research ideation, polish writing, draft sections of the paper or to generate synthetic data (data is generated from code). We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Algorithm 1 Data Preparation and Normalization

Require: U^{all}, set of all users; S^{all}, set of all sessions
Ensure: $\hat{C}_1(u_i)_k$, unit normalized value of tool k for user u_i

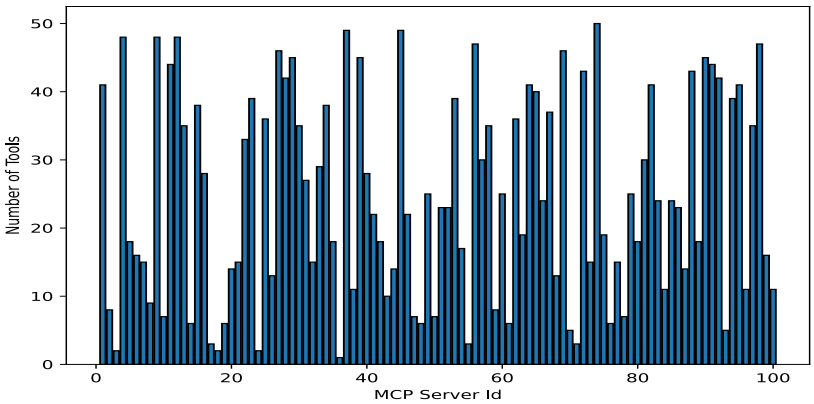
Initialize $C(u_i)_k = 0$, for all $i \in [1, |U^{all}|], k \in [1, |T^{all}|]$
Initialize $T(u_i) = \varnothing$, for all $i \in [1, |U^{all}|]$
for each $u_i \in U^{all}$ **do**
    $S_i \leftarrow \{\text{sessions of user } u_i\}$, $S_i \in S^{all}$
    for each $s \in S_i$ **do**
        for each $\tau_k \in s$ **do**, $(\tau_k \in M_k)$
            $C(u_i)_k \leftarrow C(u_i)_k + 1$
            $T(u_i) \leftarrow T(u_i) \cup \{\tau_k\}$
        end for
    end for
    for each $\tau_k \in T(u_i)$ **do**
        $\hat{C}(u_i)_k \leftarrow \dfrac{C(u_i)_k}{|T(u_i)|}$
    end for
    for each $\tau_k \in T(u_i)$ **do**
        $\hat{C}_1(u_i)_k \leftarrow \dfrac{\hat{C}(u_i)_k}{\sum_{k=1}^{|T(u_i)|} \hat{C}(u_i)_k}$
    end for
end for

Algorithm 2 Shared Autoencoder User Embeddings

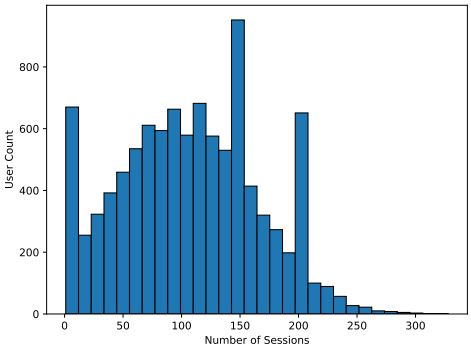
Require: $X = MAT^{u,\tau}$; embedding dimension d
Ensure: MAT^{E}; per-user reconstruction losses
Initialize shared encoder and decoder parameters with seed 1
for $epoch \leftarrow [1, epochs]$ **do**
    for each shuffled mini-batch $B \subset X$ **do**
        $E_B \leftarrow \text{Encoder}(B)$
        $\hat{B} \leftarrow \text{Decoder}(E_B)$
        $\mathcal{L}_B \leftarrow \text{WeightedMSE}(\hat{B}, B)$
        Update all model parameters using Adam and $\nabla \mathcal{L}_B$
    end for
    if $\mathcal{L}_{epoch} < 5 \times 10^{-5}$ **then**
        break
    end if
end for
$MAT^{E} \leftarrow \text{Encoder}(X)$
$loss_i \leftarrow \text{WeightedMSE}(\hat{X}_{i,:}, X_{i,:})$

Algorithm 3 Truncated-SVD User Embeddings

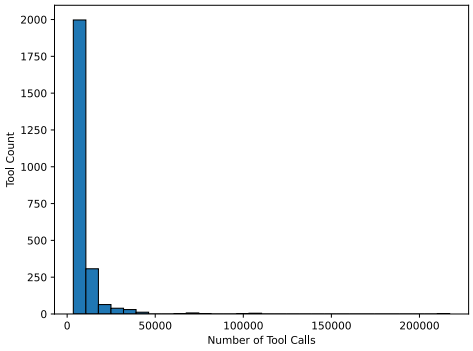
Require: $X = MAT^{u,\tau}$; embedding dimension d
Ensure: MAT^{E}; per-user reconstruction losses
$(U_d, \Sigma_d, V_d^T) \leftarrow \text{TruncatedSVD}(X, d)$
$MAT^{E} \leftarrow U_d \Sigma_d$
$\hat{X} \leftarrow MAT^{E} V_d^T$
for each $u_i \in U^{all}$ **do**
    $loss_i \leftarrow \text{MSE}(\hat{X}_{i,:}, X_{i,:})$
end for



(a) Number of tools in each MCP server

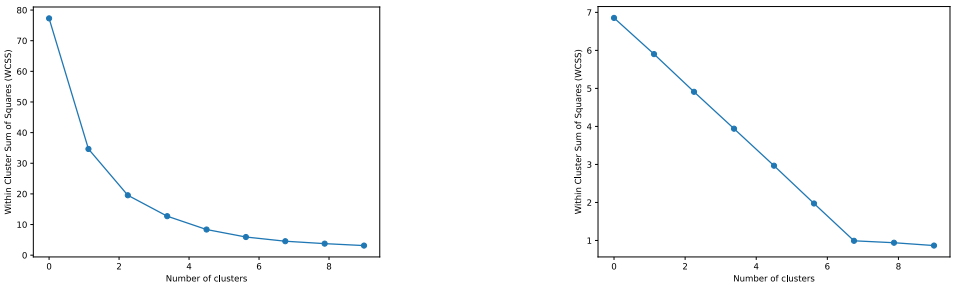


(b) Distribution: Number of sessions/user



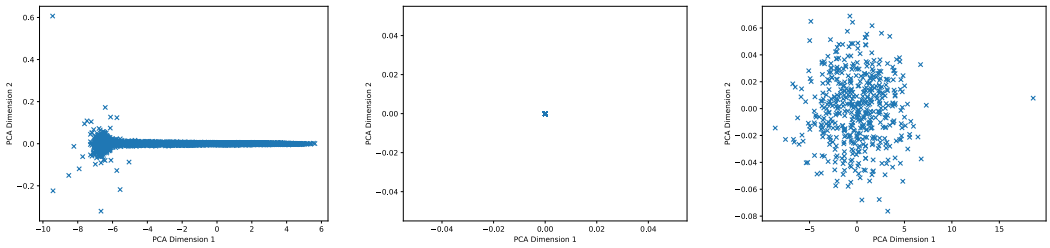
(c) Distribution: Number of tool calls

Figure A3: Summary of synthetic data.

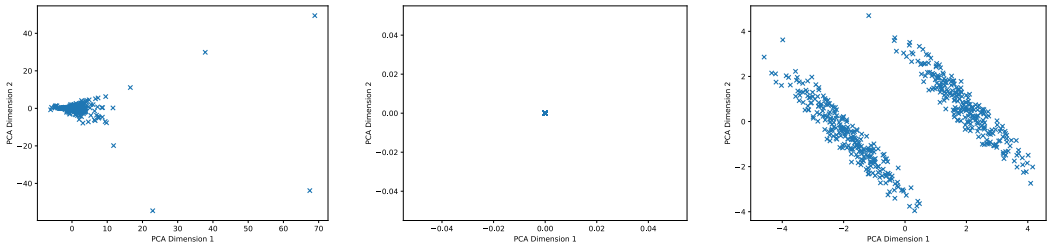


(a) Algorithm 2: 4 Optimum clusters from elbow point (b) Algorithm 3: 8 optimum clusters from elbow point

Figure A4: Embedding size 8: Number of clusters vs WCSS

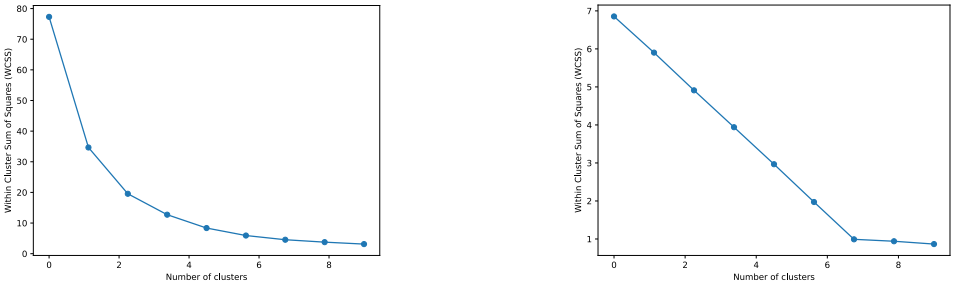


(a) Algorithm 2: All Users (b) Algorithm 2: Canary 1 Users (c) Algorithm 2: Canary 2 Users



(d) Algorithm 3: All users (e) Algorithm 3: Canary 1 Users (f) Algorithm 3: Canary 2 Users

Figure A5: Embeddings size 32: PCA for Algorithms 2 and 3



(a) Algorithm 2: 4 Optimum clusters from elbow point (b) Algorithm 3: 2 optimum clusters from elbow point

Figure A6: Embedding size 32: Number of clusters vs WCSS

Size	Distance	Algorithm	1	2	3	4	5
8	Cosine	2	2558	9075	211	5431	6059
8	Euclidean	2	9587	2815	4740	3578	1605
8	Cosine	3	7843	7844	5248	8127	2850
8	Euclidean	3	1050	7844	434	3260	7149
8	Cosine	PCA	8428	2316	5194	128	5535
8	Euclidean	PCA	8428	6415	2316	942	4765
8	Cosine	Raw	7093	1707	6631	5795	678
8	Euclidean	Raw	5828	784	4558	64	330
32	Cosine	2	3353	3219	509	6285	9591
32	Euclidean	2	2994	3253	9075	2853	4482
32	Cosine	3	6557	968	2315	7179	9447
32	Euclidean	3	6557	968	4835	1833	9447
32	Cosine	PCA	7797	2338	7235	3108	4896
32	Euclidean	PCA	9583	1538	7735	3756	3567
32	Cosine	Raw	7093	1707	6631	5795	678
32	Euclidean	Raw	5828	784	4558	64	330

Table A1: Five nearest users to non-canary user 1, sorted by increasing distance

Learned representations

Size	Group	Distance	Alg. 2		Alg. 3	
			μ	σ	μ	σ
8	Canary 1	Cosine	0	0	0	0
8	Canary 1	Euclidean	1.49×10^{-8}	0	2.11×10^{-8}	0
8	Canary 2	Cosine	7.57×10^{-6}	1.41×10^{-5}	3.26×10^{-7}	3.95×10^{-7}
8	Canary 2	Euclidean	.003047	.002419	.000691	.000418
8	All users	Cosine	.110158	.139910	.001197	.013345
8	All users	Euclidean	.376290	.280574	.024507	.042338
32	Canary 1	Cosine	0	0	0	0
32	Canary 1	Euclidean	0	0	2.07×10^{-8}	2.63×10^{-9}
32	Canary 2	Cosine	4.85×10^{-6}	8.88×10^{-6}	2.26×10^{-7}	1.84×10^{-7}
32	Canary 2	Euclidean	.002459	.001913	.000625	.000247
32	All users	Cosine	.106054	.144909	.001074	.010196
32	All users	Euclidean	.359867	.287410	.018837	.042335

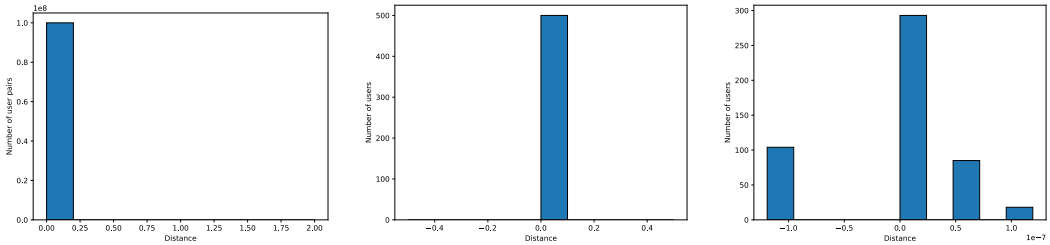
Baselines

Size	Group	Distance	PCA		Raw	
			μ	σ	μ	σ
8	Canary 1	Cosine	0	0	0	0
8	Canary 1	Euclidean	2.86×10^{-8}	3.72×10^{-8}	0	0
8	Canary 2	Cosine	5.43×10^{-5}	3.38×10^{-5}	.003933	.000355
8	Canary 2	Euclidean	.010018	.002878	.088601	.004010
8	All users	Cosine	.950836	.721049	.455178	.193707
8	All users	Euclidean	1.213140	.655717	.933881	.195508
32	Canary 1	Cosine	0	0	0	0
32	Canary 1	Euclidean	1.44×10^{-7}	1.55×10^{-7}	0	0
32	Canary 2	Cosine	.000212	6.04×10^{-5}	.003933	.000355
32	Canary 2	Euclidean	.020393	.002866	.088601	.004010
32	All users	Cosine	.969709	.606560	.455178	.193707
32	All users	Euclidean	1.289030	.527086	.933881	.195508

Table A2: Pairwise distances after per-user L2 normalization

Size	**Algorithm**	**Optimal Clusters**	Silhouette Score
8	2	4	0.562
8	3	8	0.990
8	PCA	5	0.543
8	Raw	5	0.19
32	2	4	0.560
32	3	2	0.983
32	PCA	5	0.450
32	Raw	5	0.19

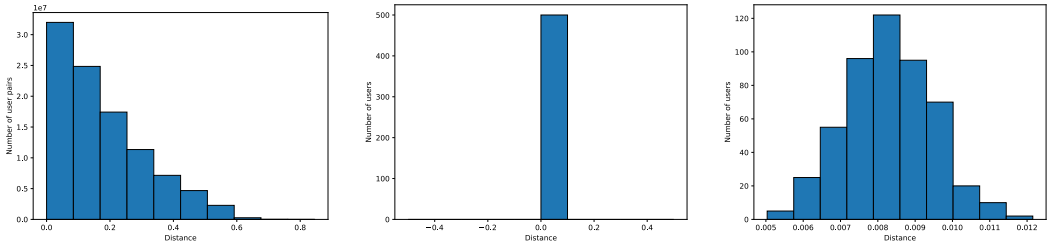
Table A3: Optimum Clusters with Silhouette Score



(a) Cosine distance between all users

(b) Cosine distance between Canary 1 users

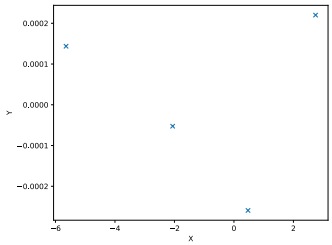
(c) Cosine distance between Canary 2 users



(d) Euclidean distance between all users

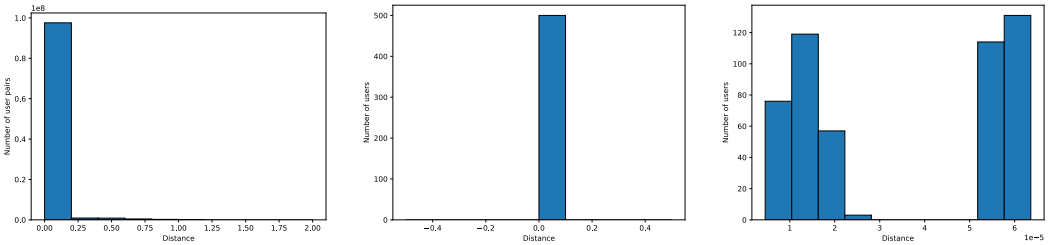
(e) Euclidean distance between Canary 1 users

(f) Euclidean distance between Canary 2 users



(g) Cluster Centroids with optimal cluster count of 4
(reduced to 2 dimensions with PCA)

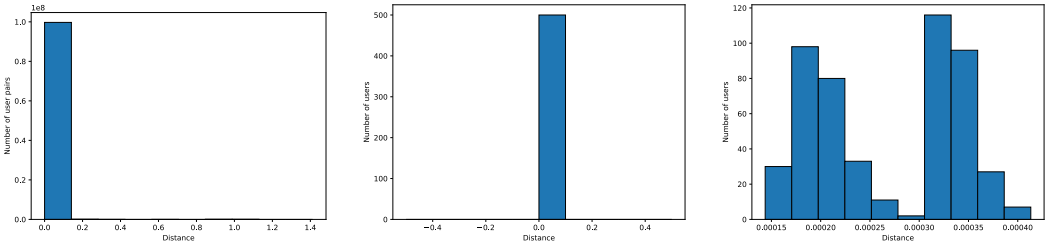
Figure A7: Embeddings-8: Results from Algorithm 2



(a) Cosine distance between all users

(b) Cosine distance between Canary 1 users

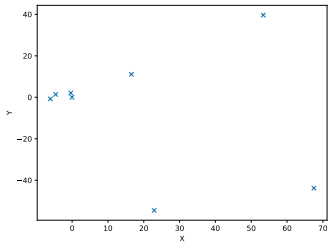
(c) Cosine distance between Canary 2 users



(d) Euclidean distance between all users

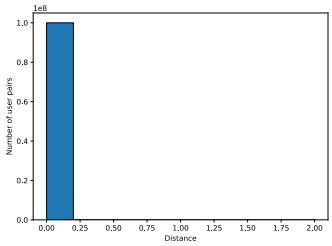
(e) Euclidean distance between Canary 1 users

(f) Euclidean distance between Canary 2 users

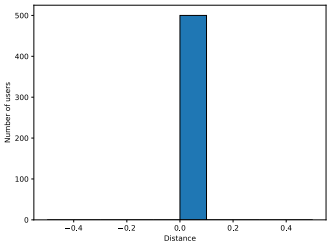


(g) Cluster centroids with optimal cluster count of 8
(reduced to 2 dimensions with PCA)

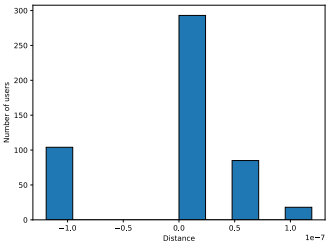
Figure A8: Embeddings-8: Results from Algorithm 3



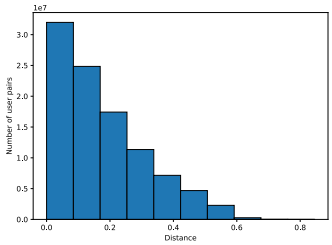
(a) Cosine distance between all users



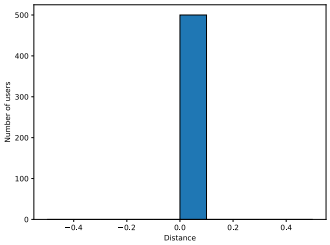
(b) Cosine distance between Canary 1 users



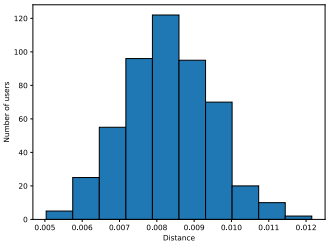
(c) Cosine distance between Canary 2 users



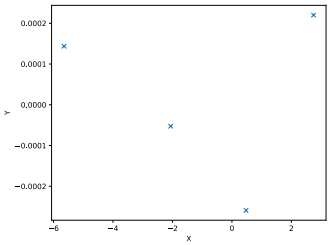
(d) Euclidean distance between all users



(e) Euclidean distance between Canary 1 users

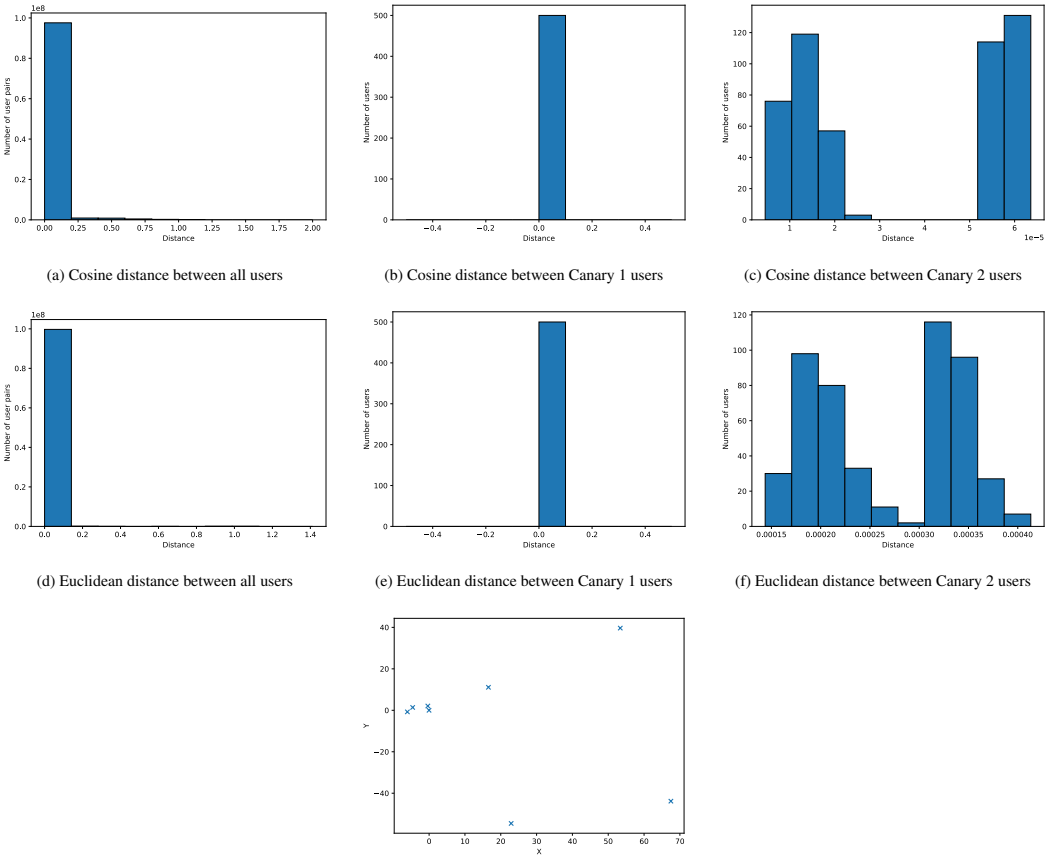


(f) Euclidean distance between Canary 2 users



(g) Cluster centroids with optimal cluster count of 4
(reduced to 2 dimensions with PCA)

Figure A9: Embeddings-32: Results from Algorithm 2



(a) Cosine distance between all users

(b) Cosine distance between Canary 1 users

(c) Cosine distance between Canary 2 users

(d) Euclidean distance between all users

(e) Euclidean distance between Canary 1 users

(f) Euclidean distance between Canary 2 users

(g) Cluster centroids with optimal cluster count of 2 (reduced to 2 dimensions with PCA)

Figure A10: Embeddings-32: Results from Algorithm 3