

Learning User Embeddings from Model Context Protocol Tool Interactions

Author information scrubbed for double-blind reviewing

No Institute Given

Abstract. As language models get increasingly used in production systems, delivering personalized experiences or determining usage patterns at scale remains a practical challenge. Most existing approaches rely on storing user text histories, maintaining explicit profiles, or fine-tuning model parameters that introduce privacy risks, engineering complexity, and high computational cost. The Model Context Protocol (MCP) offers a new opportunity: every tool invocation made by a language model provides a structured, auditable signal of user intent and workflow preferences.

In this paper, we present a lightweight framework for generating user embeddings directly from MCP tool invocation logs. Our approach uses only aggregated tool usage statistics—without storing natural language interactions or personally identifiable information—making it suitable for privacy-sensitive deployments. We introduce a normalization pipeline for tool activity data and propose two practical embedding methods: a neural-network-based factorization model and a polynomial regression approach. Both methods produce compact user representations that can be computed efficiently and updated incrementally. Using a large-scale synthetic dataset of 10,000 users interacting with 100 MCP servers, we demonstrate that the resulting embeddings support user clustering, similarity search, and behavioral segmentation. Compared to PCA and raw usage-distance baselines, our methods achieve stronger cluster separation and improved silhouette scores.

These results show that MCP interaction logs can serve as a scalable, low-overhead foundation for personalization features such as focused response strategies, adaptive tool routing, or user segmentation. Our framework is designed to integrate directly into production MCP-enabled systems, enabling structured-interaction-driven personalization without model re-training.

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

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 to 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 the feature of an interactive software system to customize individual user experiences, with the primary goal of improving user satisfaction, task efficiency, and overall engagement [9, 18]. Although LMs are interactive systems, 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 makes 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 or 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 understanding user behavior. 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 resources 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.

In the next sub-section, we give more details on MCP followed by a description of user embeddings and how they are useful. Section 2 shows the related work on LM personalization, in section 3 we mention the concepts and assumptions in our approach followed by formal definitions. Section 4 provided details of our algorithms, followed by experimental results (Section 5) that apply those algorithms. We conclude with a summary and opportunities for future work (Section 6).

1.1 Model Context Protocol

The Model Context Protocol (MCP) [1] 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 the latest authoritative and 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, allowing MCP developers to add or update tools independently of the model’s training process, thus extending the effective capabilities of the model without expensive retraining or fine-tuning.

MCP Servers can be local using the *stdio* protocol, or are accessible over the network with *StreamableHTTP*. The metadata for tools available in a server are generally fetched at run time using a MCP ‘tools/list’ call. 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 meta-data 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 which is "San Francisco". The LM then analyzes all the MCP tools available to it, looking at 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 other information such as annotation hints. Currently there are 4 such annotations (*readOnlyHint*, *destructiveHint*, *idempotentHint*, *openWorldHint*) and these tell the LM what kind of action or state changes the tool can do. The MCP protocol is undergoing continuous enhancements, and a deep dive on MCP is outside the purview of this paper and readers are guided to formal documentation [1].

1.2 User Embeddings

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 usage 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 features that can be transformed to numeric representations have to be available. Previously 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. [23] and Tseng et al. [20] 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[17] adopts the LoRA [7] method to tune a Llama model [19] for each user, and its efficiency is further improved in [16]. Zhang et al. [22] apply PEFT in a memory injected approach. Hydra [24] is a learning-based framework that captures user-specific behavior using a re-ranker.

Salemi et al. [15] explore different retrieval models and their efficiency in personalization. Richardson et al. [14] present a summary-augmented approach, and enhanced retrieval with user interest journey is presented in Christakopoulou et al. [5], while Liu et al. [10] 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. [4] who use a variational auto-encoder to learn from dialogue history, and Wu et al. [21] 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. [13] develop a novel approach of using user’s interaction history into dense embeddings which are then integrated directly into a LM 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, we do not expect MCP tools to have any defined features and use an approach of using an all-ones matrix to build the user-embeddings. Liu et al. [11] 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 [15]. Doddapaneni et al. [6] 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 reduces privacy related risks.

2.3 Matrix Factorization and Raw Distance measures

Matrix Factorization is a widely used technique in recommendation systems [8]. In general, with Matrix factorization we get two lower dimensional matrices (q and q) each revealing latent factors. If p is considered as user embeddings then q gives latent features of MCP tools, which we not want to account for as currently MCP tools do not have any classifications (like a movie can be classified with features like genre, language, etc.). Since q determines the value of p we need a different approach to determine only p .

Principal Component Analysis (PCA) [12] is another application of matrix factorization primarily applied to dimension reduction. PCA is sensitive to feature scaling where large numerical ranges dominate the components, and this will be the case with MCP as select few tools will be called much more frequently. PCA has another practical downside where the embeddings of a user cannot be computed in isolation, but will have to be recomputed for every user even if a single user has a change in tool count. Another approach is computing the distances (cosine or euclidean) of raw tool counts, but this has an embedding size equal to the number of tools and is not dense. In our experimentation, we have used PCA and raw distances for baseline measures.

3 Basic Concepts and Formal Definitions

We introduce the important concepts and notation 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 an 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 its 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, and each user has a unique identity with no assumed upper bound on their cardinality.

3.2 Assumptions

While LMs can call more than one tool per prompt, our current work assumes a single tool call per prompt. In addition, 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 is assigned 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 back 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 data threshold is implemented all new users will get similar embeddings. This will inhibit personalized experiences for new users, and mitigating this is scope for future work.

3.3 Formal Definitions

- Let LM represent a small or large language model capable of executing MCP tools.
- Let $M^{all} = \{M_{null}, M_1, M_2, M_3 \dots M_m\}$ 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.
- $T_i = \{\tau_1^i, \tau_2^i, \dots, \tau_{|T_i|}^i\}$ 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.
- $I_i^j = \{in_1, in_2, \dots, in_n\}$ be the inputs required for tool $\tau_i^j, |I_i^j| \geq 0$.
- $U^{all} = \{u_1, u_2, \dots, u_s\}$ be the set of users in LM, $|U^{all}| > 0$
- $S_i^n = \{\tau_1^m, \tau_2^m, \dots, \tau_t^m, \tau_{t+1}^m, \tau_t^m\}$ 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 $\mid \tau_t^m \in T_m, T_m \subset M_i, 0 < m \leq |M^{all}|$.
- 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 sessions in LM. Hence, $S_i \in S^{all} \mid 0 < i < |U^{all}|$.
- $E_i = (e_1^i, e_2^i, \dots, e_d^i) \in R^d$ is the embedding vector for the user $u_i \in |U^{all}|$. d is the dimensionality of the vector embeddings.

4 Algorithms

The goal of our algorithms is to generate embeddings for each user based on the available data of MCP tool calls. We have broken our approach into two parts, one to prepare and normalize the data (Algorithm 1) and others to build the embeddings (Algorithms 2 and 3). Algorithm 1 generates unit normalized $\hat{C}_1(u_i)_k$ values which are used to generate the embeddings. For each user, the normalization steps first divides the tool count by total tool calls for a user followed by a unit normalization step, this ensures a bounded range of tools calls across all users.

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 $|U^{all}| \times |T^{all}|$. The following is a representation of this matrix (referred to as $MAT^{u,\tau}$),

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

- 1: Initialize $C(u_i)_k = 0$, for all $i \in [1, |U^{all}|]$, $k \in [1, |T^{all}|]$
- 2: Initialize $T(u_i) = \emptyset$, for all $i \in [1, |U^{all}|]$
- 3: **for each** $u_i \in U^{all}$ **do**
- 4: $S_i \leftarrow \{\text{sessions of user } u_i\}$, $S_i \in S^{all}$
- 5: **for each** $s \in S_i$ **do**
- 6: **for each** $\tau_k \in s$ **do**, ($\tau_k \in M_k$)
- 7: $C(u_i)_k \leftarrow C(u_i)_k + 1$
- 8: $T(u_i) \leftarrow T(u_i) \cup \{\tau_k\}$
- 9: **end for**
- 10: **end for**
- 11: **for each** $\tau_k \in T(u_i)$ **do**
- 12: $\hat{C}(u_i)_k \leftarrow \frac{C(u_i)_k}{|T(u_i)|}$
- 13: **end for**
- 14: **for each** $\tau_k \in T(u_i)$ **do**
- 15: $\hat{C}_1(u_i)_k \leftarrow \frac{\hat{C}(u_i)_k}{\sum_{k=1}^{|T(u_i)|} \hat{C}(u_i)_k}$
- 16: **end for**
- 17: **end for**

$$MAT^{u,\tau} =$$

	τ_1^m	τ_2^m	...	$\tau_{ T }^m$
u_1	$\hat{C}_1(1)_1$	$\hat{C}_1(1)_2$...	$\hat{C}_1(1)_{ T }$
u_2	$\hat{C}_1(2)_1$	$\hat{C}_1(2)_2$...	$\hat{C}_1(2)_{ T }$
u_3	$\hat{C}_1(3)_1$	$\hat{C}_1(3)_2$...	$\hat{C}_1(3)_{ T }$
...
$u_{ U -1}$	$\hat{C}_1(U -1)_1$	$\hat{C}_1(U -1)_2$...	$\hat{C}_1(U -1)_{ T }$
$u_{ U }$	$\hat{C}_1(U)_1$	$\hat{C}_1(U)_2$...	$\hat{C}_1(U)_{ T }$

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 .

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

Our goal is to compute the unknown values in MAT^E using the known values of $MAT^{u,\tau}$, or we need to determine an operation f_{op} :

$$f_{op} : MAT^E \rightarrow MAT^{u,\tau}$$

f_{op} does a linear transformation of MAT^E to $MAT^{u,\tau}$ which can be achieved with a matrix multiplication operation with another matrix MAT^x :

$$MAT^E \times MAT^x \approx MAT^{u,\tau} \quad (1)$$

where,

$$MAT^E \in R^{|U| \times d},$$

$$MAT^{u,\tau} \in R^{|U| \times |T|}.$$

To allow this matrix operation, the size of MAT^x is:

$$MAT^x \in R^{d \times |T|}$$

Since MAT^x is required only to complete the matrix operation we can use an all-ones matrix, or:

$$MAT^x =$$

	1	2	...	$ T - 1$	$ T $
1	1	1	...	1	1
2	1	1	...	1	1
...
$d - 1$	1	1	...	1	1
d	1	1	...	1	1

One will observe that eq.(1) is similar to matrix factorization into two matrices p and q . where $p = MAT^x$ and q is the all-ones matrix. As mentioned earlier, currently q does not represent any latent features of tools so it is a constant and needed only to complete the matrix operation.

The unknown values are in MAT^E and the approximation of these values can be computed through an optimization process similar to what is done for weights in a feed-forward neural network. To further elaborate, the output of a feed-forward neural network layer is given by:

$$z = W^T \cdot x + b \quad (2)$$

when $b = 0$, $W^T = MAT^E$, $z = MAT^{u,\tau}$, and x being one row from MAT^x then eq.(1) and (2) are similar allowing us to derive MAT^E using optimization and backpropagation as implemented in a feed-forward network. This allows MAT^E to be approximated using any of the common neural network libraries like TensorFlow, PyTorch etc. and steps of this process are given in Algorithm 2.

The cost function for the optimization process is Mean Square Error (MSE) represented by MSE , as MSE is well-suited for regression. $rand(s)$ represents random number with seed s . $MAT_{i,:}^E$ represents the embedding vector of user u_i , while $MAT_{i,:}^{loss}$ contains the values to update the embeddings(weights) using partial derivatives in the backpropagation step.

Algorithm 2 Compute User Embeddings using Neural Network

Require: $MAT^{u,\tau}$, matrix of tools for each user
Ensure: MAT^E , embedding for each user
1: Initialize $MAT^x \in R^{d \times |T|}$ with all 1's
2: $MAT^E \in R^{|U| \times d} \leftarrow rand(s)$
3: **for each** $u_i \in U^{all}$ **do**
4: **for each** $epoch \in [1, epochs]$ **do**
5: $z_i \leftarrow MAT_{i,:}^E \times MAT^x$
6: $loss \leftarrow MSE(z_i, MAT_{i,:}^{u,\tau})$
7: $MAT_{i,:}^{loss} \leftarrow backpropagate(loss)$
8: $MAT_{i,:}^E \leftarrow optimizer_update(MAT_{i,:}^{loss})$
9: **end for**
10: **end for**

Another method (Algorithm 3) to compute embeddings is by determining a polynomial that *attempts* to fit the maximum number of points, where the points are the number of calls for each tool and the order of the polynomial equals the embedding dimensions. With this approach, each row in $MAT^{u,\tau}$ is passed to a function which generates a unique polynomial fit for each user, and the polynomial coefficients determine the embeddings.

The generic polynomial equation p^i for given embedding dimension d for user u_i is given below:

$$p^i = a_{d-1}.x^{d-1} + a_{d-2}.x^{d-2} + \dots + a_1.x^1 + a_0.x^0$$

The coefficients of this polynomial are solved using singular value decomposition (SVD) with the goal of minimizing the sum of squared residuals (method of least squares). In general, for determining the coefficients the following equation is applied for each tool τ_j^m :

$$\hat{C}_1(i)_j = a_{d-1}.j^{d-1} + a_{d-2}.j^{d-2} + \dots + a_1.j^1 + a_0.x^0$$

j is the index of the tool ($j \in [0, |T^{all}|]$), and $\hat{C}_1(i)_j$ is the normalized value of the tool call count.

The coefficients of the final fit polynomial, map to embeddings of user u_i :

$$e_j^i = a_{j-1}, j \in [d, 1], a_j \in R, e_j^i \in MAT^E$$

The *poly_fit* function in Algorithm 3 is an abstraction of polynomial fit using the least squares method. $MAT_{i,:}^E$ represents the embedding vector of user u_i .

4.1 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:

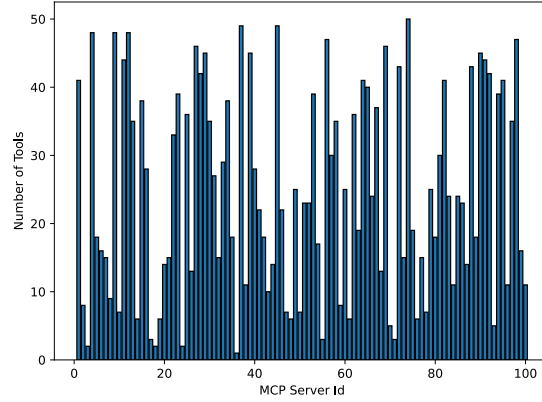
Algorithm 3 Compute User Embeddings using Polynomial Fit

Require: $MAT^{u,\tau}$, matrix of tools for each user

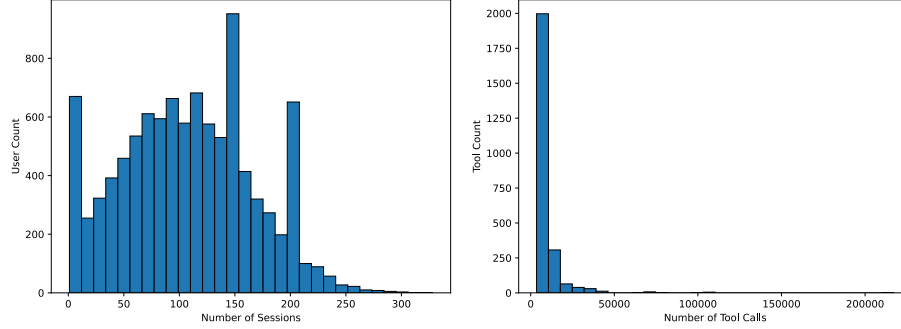
Ensure: MAT^E , embedding for each user

```

1: for each  $u_i \in U^{all}$  do
2:    $(coeffs_i, residuals_i) \leftarrow \text{poly\_fit}(MAT_{i,:}^{u,\tau})$ 
3:    $loss_i \leftarrow residuals_i$ 
4:    $MAT_{i,:}^E \leftarrow coeffs_i$ 
5: end for
  
```



(a) Number of tools in each MCP server



(b) Distribution: Number of sessions/user (c) Distribution: Number of tool calls

Fig. 1: Summary of synthetic data.

$$\cos_dist(u_i, u_j) = 1 - \frac{(MAT_{i,:}^E) \cdot (MAT_{j,:}^E)}{\|MAT_{i,:}^E\| \|MAT_{j,:}^E\|},$$

$$\cos_dist : R^n \times R^n \rightarrow [0, 2].$$

and,

$$euc_dist(u_i, u_j) = \sqrt{\sum_{i=1}^{|T^{all}|} (MAT_{i,:}^E - MAT_{j,:}^E)^2},$$

$$euc_dist : R^n \times R^n \rightarrow [0, \inf)$$

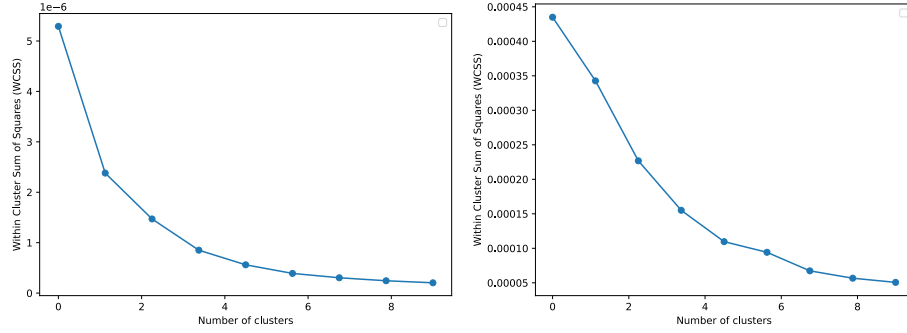
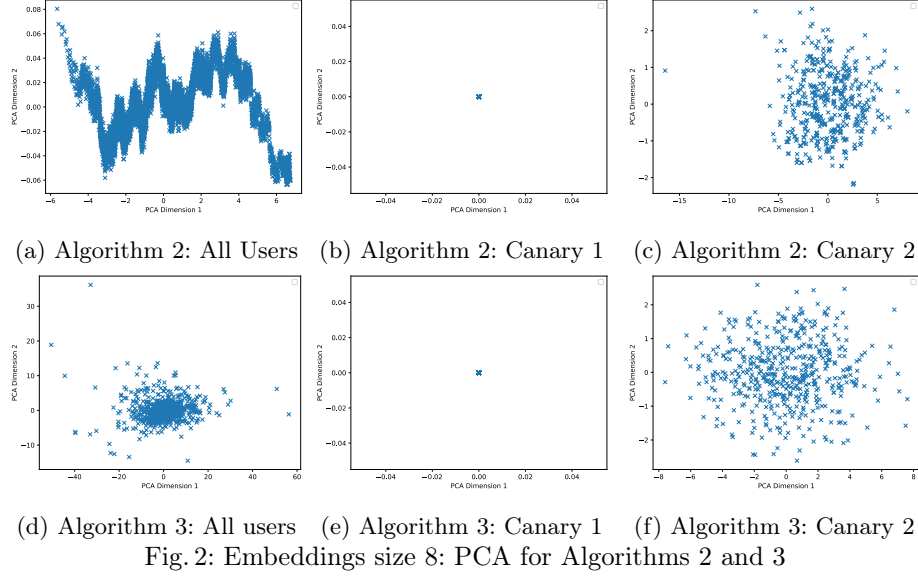
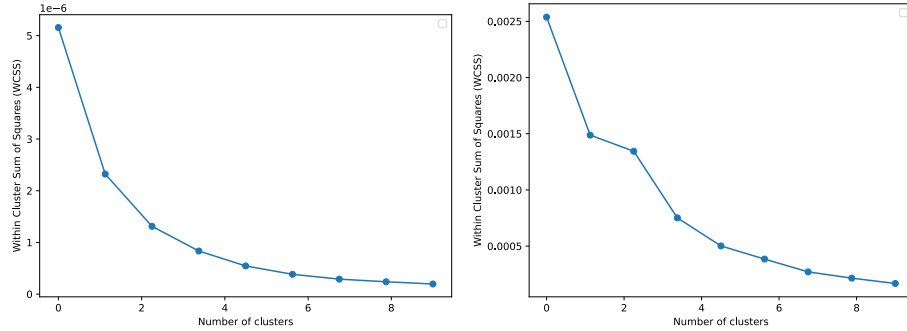
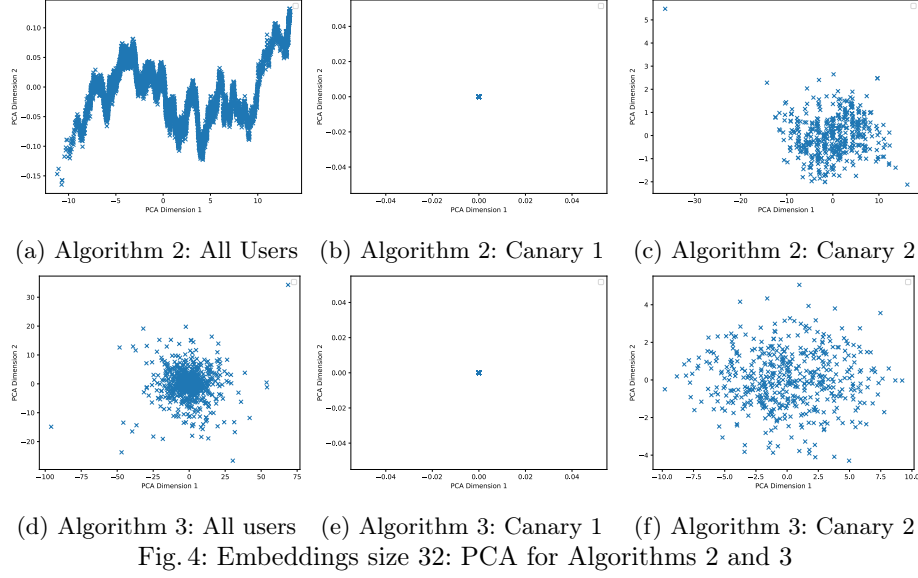


Fig. 3: Embedding size 8: Number of clusters vs WCSS

5 Experimental Results

Currently commercially available MCP servers have proprietary data which we cannot disclose in this paper, and existing personalization benchmarks like LaMP [15] do not provide MCP tool invocation data. Hence, for evaluation of our methods we have generated new synthetic data in a SQLite database which has also been shared for wider use. 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 [2], while the SQLite database with the generated synthetic data is available at [3].

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. At depth d , the probability of selecting a tool from the 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 as the current prompt.

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 that the sample has known attributes that helps us determine the effectiveness of user embeddings. Data of 10% of randomly selected user's are modified this way and we classify these samples as canary users, which are further sub-divided in two types.

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.

Canary Users 2 Composing the other 5% of the users, all users in this group have 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

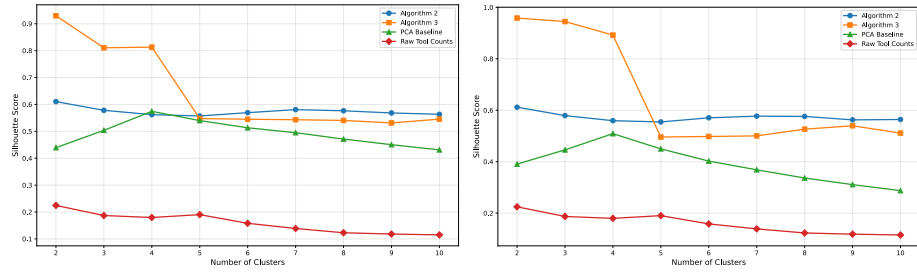
In Figure 1, the main characteristics of the synthetic data are shown. These are the number of tools in each MCP server and the distribution of sessions and tool calls. From Fig. 1(c), we observe a skew with some tools used more frequently than others. This reflects practical scenarios where tools for common functions (e.g. summarization) will be used more often than others.

Results for embedding dimensions of 8 and 32 are shown separately. Using PCA we have transformed the embeddings to 2 principal components, these are shown in figs. 2 and 4 for embedding dimensions of 8 and 32, respectively. Additional result plots are provided in the Appendix.

The principal components for Canary 1 users are same for both algorithms showing up as a single point in Figs.2b/2e and 4b/4e . Compared to the PCA of all users (2a/2d, 4a/4d), the PCA for Canary 2 users (2c/2f, 4c/4f) show more closeness between embeddings. Both the canary 1 and canary 2 results are on expected lines.

Clustering and topK users In personalization, knowing the distinct clusters of users helps determine distinct user attributes like persona, corporate roles etc. Using the elbow method (Within Cluster Sum of Squares - WCSS) we have determined an optimum number of clusters (one with lowest WCSS) for embedding size of 8 (3a/3b) and 32 (5a/5b).

Finding the topK users closest to a given user also has several use cases for personalization purposes. In table 1 the topK (k=5) users determined for a randomly selected user id $u_i = 1$, where u_i is not in any canary group are shown. Apart from a single user id (6004) for euclidean distance, the topK users determined from each algorithm are different.



(a) Embeddings-8 (b) Embeddings-32
Fig. 6: Silhouette scores for different cluster sizes

5.3 Baseline Comparisons

In related work, we mentioned that PCA and raw distance measurements can also be treated as embeddings. We intentionally focus on these simple baselines to isolate the value of MCP tool semantics rather than focus model complexity. The average distance and cluster metrics with these baselines have been compared with our algorithms. When computing the PCA and raw distance measurements we have taken the actual tool counts without normalizing them via algorithm 1.

Distance Measures For Canary 1, canary 2 and all users we have computed the distance between each pair and taken the average (μ) and standard deviation (σ) of the distances. Since the PCA and raw distance values are not normalized, for a comparative scale the embeddings have been L2 normalized before computing their distance metrics (Table 2).

For Canary 1 the μ and σ are 0 except for euclidean distance for PCA. We have verified this is not due to rounding and is likely due to orthogonal projection, but further work is needed to confirm this. For canary 2 and all users, all approaches show different divergence in pair-wise distances and there is no significant distinction when embeddings sizes vary from 8 to 32.

Clustering and topK users Table 3 shows the optimal number of clusters (derived from elbow method) and the silhouette score of the optimal clusters for each algorithm. The silhouette scores are in $[1, -1]$ where a higher score denotes well separated clusters. We have also computed Silhouette scores as number of clusters increase from 1 to 10, these are shown in figure 6 for each algorithm.

From table 3 and figure 6, it is clear that both algorithm 2 and 3 have better (higher) silhouette scores than the baselines. Algorithm 2 and 3 also have similar scores, though the optimal cluster count from algorithm 2 is smaller than the rest.

6 Conclusions and Future Work

In this paper, we have presented methods to generate low-dimension user embeddings from MCP interactions and, using synthetic data, we show that using distance measures these embeddings can determine similar users as well as iden-

Size	Distance	Algorithm	1	2	3	4	5
8	Cosine	2	554	584	1691	5460	6336
8	Euclidean	2	3116	6004	7147	7590	9842
8	Cosine	3	7147	1851	3827	763	9740
8	Euclidean	3	6004	4338	580	6920	5454
8	Cosine	PCA	9258	9064	1341	9972	3505
8	Euclidean	PCA	46	59	80	95	201
32	Cosine	2	4111	4655	6187	8	72
32	Euclidean	2	9842	7147	6947	3116	6004
32	Cosine	3	9834	7732	6694	4395	806
32	Euclidean	3	4338	6004	5902	580	6154
32	Cosine	PCA	2340	6311	9113	5048	1670
32	Euclidean	PCA	8530	5795	46	59	80
-	Cosine	Raw	6014	6947	7185	7372	8593
-	Euclidean	Raw	5795	9150	8530	4768	7950

Table 1: topK(k=5) users for non-canary user id 1, sorted in decreasing distance measurement value

tify unique user clusters. The first algorithm prepares and normalizes the data, following which we presented two different methods for embedding generation.

Our experiments on synthetic data on the algorithms show that the embeddings for all users who make the same tool calls (Canary 1) is exact, while the embeddings for users with small differences in tool calls have corresponding closer embeddings. The embeddings for all users are distinct as seen by 2-dimensional PCA reduction. Comparisons with baseline approaches add further context to the performance of both algorithms. Using different sizes for the dense embeddings we showed that a smaller embedding size (8) is enough to distinguish the users, though further work is required to determine optimal embedding dimensions as number of users increase.

Additionally the synthetic data used for this has been shared allowing other researchers to utilize the data in their work.

6.1 Future Work

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

- Our current algorithms assume a single tool call per prompt, but LMs can invoke multiple tools which will require changes in the algorithms.
- Apart from absolute tool count, considering the temporal ordering of tool calls will lead to more representative embeddings.
- Current methods can also be used to characterize autonomous agents using embeddings based on tool calls. Our initial work on this has shown that this could be a promising approach in distinguishing agents and monitoring their behavior.

Size	Group	Distance	Alg. 2		Alg. 3		PCA		Raw	
			μ	σ	μ	σ	μ	σ	μ	σ
8	Canary 1	Cosine	0	0	0	0	0	0	0	0
8	Canary 1	Euclidean	0	0	0	0	.036	.043	0	0
8	Canary 2	Cosine	8	7	1	2	54	34	3933	355
8	Canary 2	Euclidean	3748	1560	1433	891	10018	2878	88601	4010
8	All Users	Cosine	410	14142	2014	11829	9508	721049	455178	193707
8	All Users	Euclidean	16927	18289	33747	53757	1313140	655717	933881	195508
32	Canary 1	Cosine	0	0	0	0	.029	.03	0	0
32	Canary 1	Euclidean	0	0	0	0	.013	.016	0	0
32	Canary 2	Cosine	5	3	.86	1	212	60	3933	355
32	Canary 2	Euclidean	3001	940	10	4731	20393	2866	88601	4010
32	All Users	Cosine	341	14140	1162	8325	969709	606560	455178	193707
32	All Users	Euclidean	14418	16567	24074	41766	933881	195508	933881	195508

Table 2: Distance baseline comparisons (all values are e-6)

Size	Algorithm	Optimal Clusters	Silhouette Score
8	2	4	0.562
8	3	5	0.545
8	PCA	5	0.54
32	2	4	0.55
32	3	5	0.5
32	PCA	5	0.45
-	Raw	5	0.19

Table 3: Optimum Clusters with Silhouette Score

- 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.

References

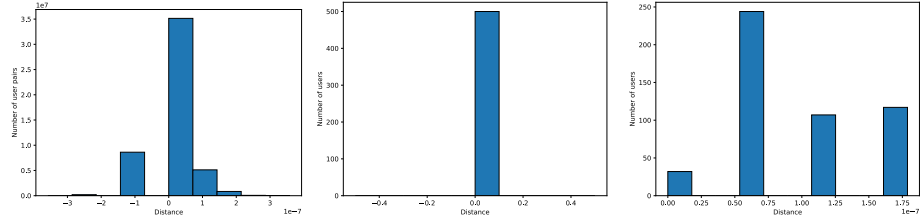
1. Model context protocol. <https://modelcontextprotocol.io/> (2024), (Accessed: Sept. 22, 2025)
2. Experiments code repository. <https://tinyurl.com/emb-repo> (2026), (Accessed: Jan. 15, 2026)
3. Shared files. <https://tinyurl.com/emd-share-1> (2026), (Accessed: Jan. 15, 2026)
4. Cho, I., Wang, D., Takahashi, R., Saito, H.: A personalized dialogue generator with implicit user persona detection (2022), <https://arxiv.org/abs/2204.07372>
5. Christakopoulou, K., Lalama, A., Adams, C., Qu, I., Amir, Y., Chuceri, S., Vollucci, P., Soldo, F., Bseiso, D., Scodel, S., Dixon, L., Chi, E.H., Chen, M.: Large language models for user interest journeys (2023), <https://arxiv.org/abs/2305.15498>
6. Doddapaneni, S., Sayana, K., Jash, A., Sodhi, S., Kuzmin, D.: User embedding model for personalized language prompting (2024),

<https://arxiv.org/abs/2401.04858>

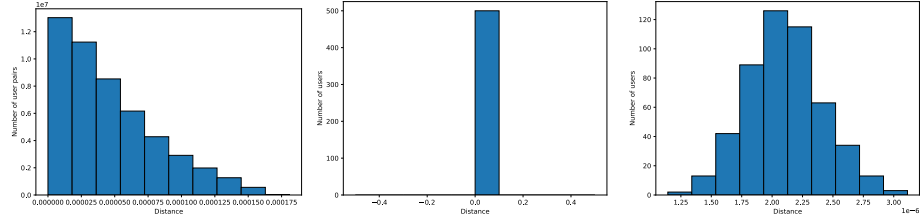
7. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022), <https://openreview.net/forum?id=nZeVKeeFYf9>
8. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. *Computer* **42**(8), 30–37 (2009)
9. Li, S., Karahanna, E.: Personalized content recommendation and user satisfaction: Theoretical synthesis and empirical findings. *Journal of Management Information Systems* **23**(3), 45–70 (2007)
10. Liu, J., Jin, J., Wang, Z., Cheng, J., Dou, Z., Wen, J.R.: Reta-llm: A retrieval-augmented large language model toolkit (2023), <https://arxiv.org/abs/2306.05212>
11. Liu, J., Zhu, Y., Wang, S., Wei, X., Min, E., Lu, Y., Wang, S., Yin, D., Dou, Z.: Llms + persona-plug = personalized llms (2024), <https://arxiv.org/abs/2409.11901>
12. Maćkiewicz, A., Ratajczak, W.: Principal components analysis (pca). *Computers Geosciences* **19**(3), 303–342 (1993)
13. Ning, L., Liu, L., Wu, J., Wu, N., Berlowitz, D., Prakash, S., Green, B., O’Banion, S., Xie, J.: User-llm: Efficient llm contextualization with user embeddings (2024), <https://arxiv.org/abs/2402.13598>
14. Richardson, C., Zhang, Y., Gillespie, K., Kar, S., Singh, A., Raeesy, Z., Khan, O.Z., Sethy, A.: Integrating summarization and retrieval for enhanced personalization via large language models (2023), <https://arxiv.org/abs/2310.20081>
15. Salemi, A., Mysore, S., Bendersky, M., Zamani, H.: Lamp: When large language models meet personalization (2024), <https://arxiv.org/abs/2304.11406>
16. Tan, Z., Liu, Z., Jiang, M.: Personalized pieces: Efficient personalized large language models through collaborative efforts. *CoRR* **abs/2406.10471** (2024). <https://doi.org/10.48550/ARXIV.2406.10471>, <https://doi.org/10.48550/arXiv.2406.10471>
17. Tan, Z., Zeng, Q., Tian, Y., Liu, Z., Yin, B., Jiang, M.: Democratizing large language models via personalized parameter-efficient fine-tuning. *CoRR* **abs/2402.04401** (2024). <https://doi.org/10.48550/ARXIV.2402.04401>, <https://doi.org/10.48550/arXiv.2402.04401>
18. Teevan, J., Dumais, S.T., Liebling, D.J.: 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)
19. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., Lample, G.: Llama: Open and efficient foundation language models. *CoRR* **abs/2302.13971** (2023). <https://doi.org/10.48550/ARXIV.2302.13971>, <https://doi.org/10.48550/arXiv.2302.13971>
20. Tseng, Y.M., Huang, Y.C., Hsiao, T.Y., Chen, W.L., Huang, C.W., Meng, Y., Chen, Y.N.: Two tales of persona in llms: A survey of role-playing and personalization (2024), <https://arxiv.org/abs/2406.01171>
21. Wu, B., Shi, Z., Rahmani, H.A., Ramineni, V., Yilmaz, E.: Understanding the role of user profile in the personalization of large language models (2024), <https://arxiv.org/abs/2406.17803>
22. Zhang, K., Qing, L., Kang, Y., Liu, X.: Personalized llm response generation with parameterized memory injection. *arXiv preprint arXiv:2404.03565* (2024)

- 23. Zhang, Z., Rossi, R.A., Kveton, B., Shao, Y., Yang, D., Zamani, H., Derroncourt, F., Barrow, J., Yu, T., Kim, S., Zhang, R., Gu, J., Derr, T., Chen, H., Wu, J., Chen, X., Wang, Z., Mitra, S., Lipka, N., Ahmed, N., Wang, Y.: Personalization of large language models: A survey (2025), <https://arxiv.org/abs/2411.00027>
- 24. Zhuang, Y., Sun, H., Yu, Y., Qiang, R., Wang, Q., Zhang, C., Dai, B.: Hydra: Model factorization framework for black-box llm personalization (2024), <https://arxiv.org/abs/2406.02888>

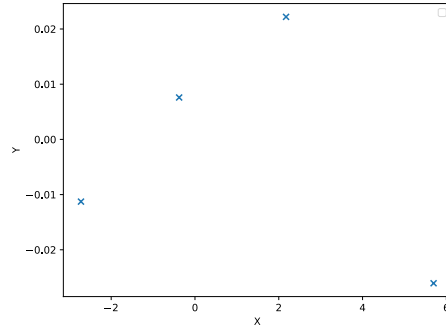
1 Additional Experiment Results



(a) Cosine distance between all users (b) Cosine distance between Canary 1 users (c) Cosine distance between Canary 2 users

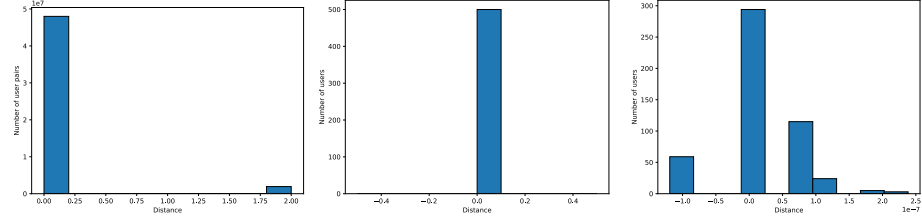


(d) Euclidean distance between all users (e) Euclidean distance between Canary 1 users (f) Euclidean distance between Canary 2 users

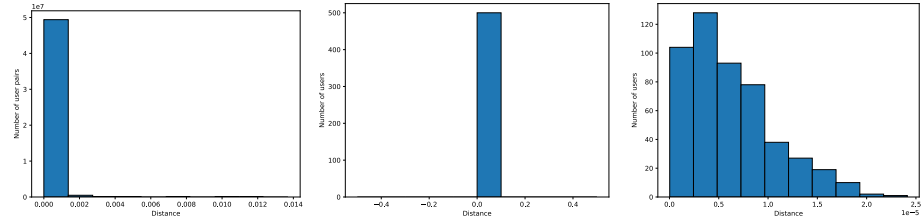


(g) Cluster Centroids with optimal cluster count of 4 (reduced to 2 dimensions with PCA)

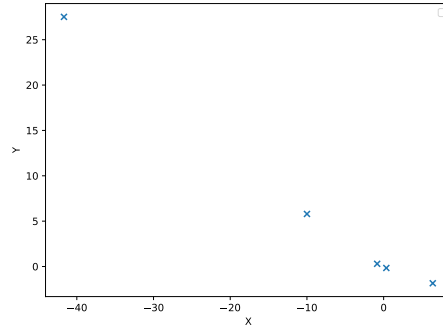
Fig. 7: Embeddings-8: Results from Algorithm 2



(a) Cosine distance between all users (b) Cosine distance between Canary 1 users (c) Cosine distance between Canary 2 users

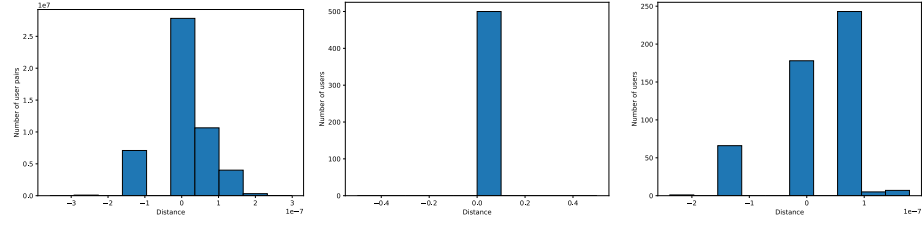


(d) Euclidean distance between all users (e) Euclidean distance between Canary 1 users (f) Euclidean distance between Canary 2 users

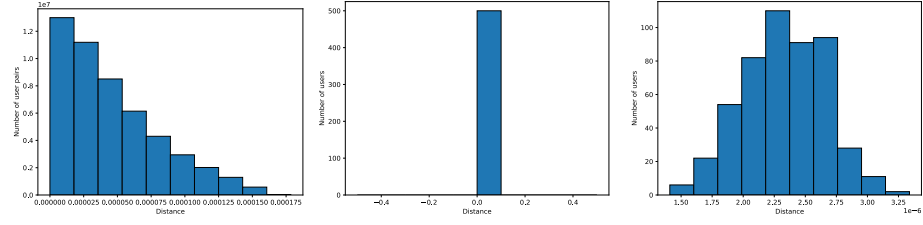


(g) Cluster Centroids with optimal cluster count of 5 (reduced to 2 dimensions with PCA)

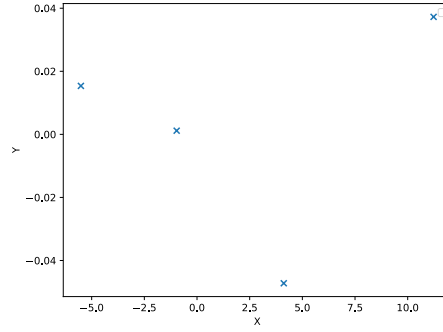
Fig. 8: Embeddings-8: Results from Algorithm 3



(a) Cosine distance between all users (b) Cosine distance between Canary 1 users (c) Cosine distance between Canary 2 users

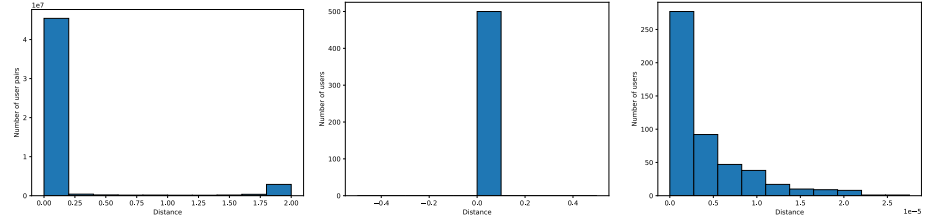


(d) Euclidean distance between all users (e) Euclidean distance between Canary 1 users (f) Euclidean distance between Canary 2 users

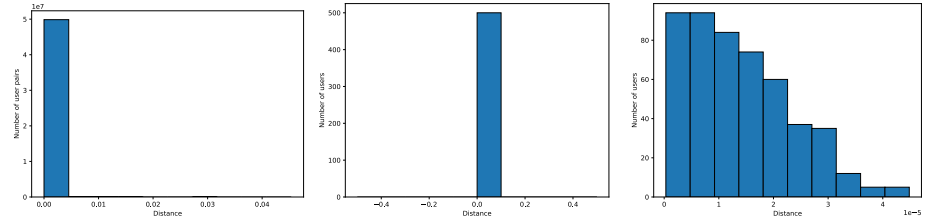


(g) Cluster Centroids with optimal cluster count of 4 (reduced to 2 dimensions with PCA)

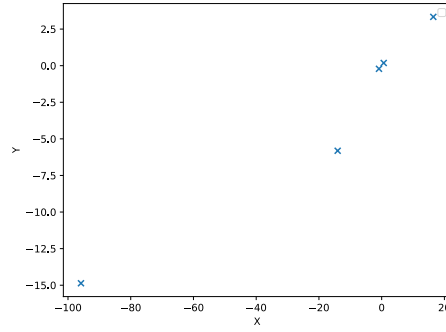
Fig. 9: Embeddings-32: Results from Algorithm 2



(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 5 (reduced to 2 dimensions with PCA)

Fig. 10: Embeddings-32: Results from Algorithm 3