Computing User Embeddings from Model Context Protocol Tool for Personalized Experiences in Language Models

Chaitanya Belwal, Ravdeep Pasricha, Vineeth Thayanithi, Gurpreet Sohal
Microsoft, USA
{cbelwal,rpasricha,vthayanithi,gurpreet.sohal}@microsoft.com

Abstract

Model Context Protocol (MCP) is a mechanism that allows Small and Large Language Models to call external data sources in real-time to 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 of the user's prompt. Due to it's discrete nature, use of MCP tools give an opportunity to understand user's behavior and allow the Language Model to build personalized experiences like initial prompt suggestions, customizing responses etc. In this paper, we propose novel approaches which use MCP tool invocation data to derive unique user embeddings which can be used to determine similarity between users of a Language Model providing a foundation for personalization related features. We present 3 algorithms, the first to normalize tool invocation data while the other two generate user embeddings using a neural network and a polynomial. We adopt a formal approach to the problem and prove our results with experimental data sets.

CCS Concepts

• **Computing methodologies** → *Information extraction*; **Machine learning approaches**; • **Information systems** → *Users and interactive retrieval.*

Keywords

Machine learning, personalization, model context protocol, language models

ACM Reference Format:
Chaitanya Belwal, Ravdeep Pasricha, Vineeth Thayanithi, Gurpreet Sohal. 2026. Computing User Embeddings from Model Context Protocol Tool for Personalized Experiences in Language Models. In . ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 a feature of an interactive software system to customize individual user experiences, with the primary goal of improving user satisfaction, task efficiency, and overall engagement [8, 16]. While LM 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 give 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 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 resources they most frequently depend upon. Unlike static preference data, tool invocation logs provide dynamic, context-sensitive insights that organically evolve 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 preserve privacy 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 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 provided details of our algorithms, followed by experimental results (Section 5) that apply the 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 authoritative and current information. There are no protocol specified limits on the number of tools each MCP serves 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.

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 agents capable of orchestrating multiple external resources through formally defined, semantically discoverable interfaces.

For selection of a tool 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 choosen 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 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 data 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 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 will enable 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 more importantly, preserving user privacy.

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 data that is available 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 data also preserves privacy. 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[21] and Tseng et al [18] provide comprehensive surveys 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[15] adopts the LoRA [7] method to tune a Llama model [17] for each user, and it's efficiency is further improved in [14]. Zhang et al[20] apply PEFT in a memory injected approach. Hydra[22] is a learning-based framework that captures user-specific behavior using a re-ranker.

Salemi et al[13] explore different retrieval models and their efficiency in personalization. Richardson et al [12] present a summary-augmented approach, and augmenting retrieval with user interest journey is presented in Christakopoulou et al[5], while Liu et al[9] 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 [19] 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 [11] 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, we do not expect MCP tools to have any defined features and use the novel approach of using a one-only matrix to build the user-embeddings.[10] 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 [13]. 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 preserves privacy.

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. *User's* are humans that interact with the LM, and each user has a unique identity with there being no limit on their cardinality.

3.2 Assumptions

While a LM can call more than one tools for each prompt, for our current work we have assumed that there is 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 has a unique id and has 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 data threshold is implemented 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 $M^{all} = \{M_{null}, M_1, M_2, M_3 \dots M_m\}$ be a set of MCP servers, that can be called from $LM \mid |M^{all}| > 0$. M_{null} is a special case where LM does not use any MCP Server when responding to a user's prompt.
- $T_i = \{\tau_1^i, \tau_2^i, \dots, \tau_o^i\}$ 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 τ_i^j, $|I_i^j| \geq 0$.
- $U^{all} = \{u_1, u_2, \dots, u_s\}$ be the set of users in LM, $|U^{all}| > 0$
- $S_i^n = \{\tau_1^x, \tau_2^x, ..., \tau_t^x, \tau_{t+1}^x, \tau_t^x\}$ 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 $\mid \tau_t^x \in T_x, T_x \subset M_i, 0 < x \leq |M^{all}|$.
- For any two $\tau_t^x, \tau_{t+n}^x \mid n > 0$ present in S_i^n, τ_t^x is called before τ_{t+n}^x.
- 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 Algorithm

The goal of the 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 other to build the embeddings (Algorithms 2 and 3). The unit normalized $\hat{C_1}(u_i)_k$ values produced by Algorithm 1 will be used to generate the embeddings.

Algorithm 1 Data Preparation

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**
  $S_i \leftarrow$ {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**
    $$\hat{C}(u_i)_k \leftarrow \frac{C(u_i)_k}{\sum_{k=1}^{|T(u_i)|} C(u_i)_k}$$
  end for
  for each $\tau_k \in T(u_i)$ **do**
    $$\hat{C_1}(u_i)_k \leftarrow \frac{\hat{C}(u_i)_k}{\sum_{k=1}^{|T(u_i)|} \hat{C}(u_i)_k}$$
  end for
end for

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

$MAT^{u,\tau}$ =

	τ_1^x	τ_2^x	...	$\tau_{\|T\|}^x$
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, $u_i \in U^{all}$, $\tau_i^x \in T^{all}$.

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

MAT^E =

u_1	e_1^1	e_2^1	...	e_d^1
u_2	e_1^2	e_2^2	...	e_d^2
u_3	e_1^3	e_2^3	...	e_d^3
...	...	...	...	...
$u_{\|U\|-1}$	$e_1^{\|U\|-1}$	$e_2^{\|U\|-1}$	...	$e_d^{\|U\|-1}$
$u_{\|U\|}$	$e_1^{\|U\|}$	$e_2^{\|U\|}$	...	$e_d^{\|U\|}$

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

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

$\oplus$ 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:

$$MAT^E \times MAT^x \approx MAT^{u,\tau} \tag{1}$$

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

Note that eq.(1) is similar to collaborative filtering where MAT^x will represent the features of the tools.However, in our work the availability of these features is not a requirement, thus an all-ones matrix is used as a substitute.

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 elaborate further, the output of a feed-forward neural network layer is given by:

$$z = W^T.x + b \tag{2}$$

when b = 0, $W^T = MAT^E$ and $z = MAT^{u,\tau}$, 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 net libraries like TensorFlow, PyTorch etc. and steps of this process are given in Algorithm 2 below.

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
Initialize $MAT^{x} \in R^{d \times |T|}$ with all 1's
$MAT^{E} \in R^{|U| \times d} \leftarrow rand(s)$
for each $u_i \in U^{all}$ **do**
  for each $epoch \in [1, epochs]$ **do**
    $z_i \leftarrow MAT^{E}_{i,:} \times MAT^{x}$
    $loss \leftarrow MSE(z_i, MAT^{u,\tau}_{i,:})$
    $MAT^{loss}_{i,:} \leftarrow backpropagate(loss)$
    $MAT^{E}_{i,:} \leftarrow optimizer\_update(MAT^{loss}_{i,:})$
  end for
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. The order of the polynomial equals the embedding dimensions and the co-efficients determine the embeddings. With this approach, each row in $MAT^{u,\tau}$ is passed to a function which uses the method of least squares to generate a unique polynomial fit for each user, coefficients for which determine the embeddings.

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

$$p^i = a_{d-1}x^{d-1} + a_{d-2}x^{d-2} + \cdots + a_1x^1 + a_0x^0$$

The coefficients of this polynomial are computed using method of least square, and x the dimensional input representing the index of tools ($x \in [0, |T^{all}|$), and the corresponding polynomial value the number of times the tool is called.

The mapping of the polynomial coefficients a map to embeddings of user u_i:

$e^i_j = a_{j-1}, j \in [d, 1], a_j \in \mathbb{R}, e^i_j \in MAT^{E}$

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

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
for each $u_i \in U^{all}$ **do**
  $(coeffs_i, residuals_i) \leftarrow$ poly_fit$(MAT^{u,\tau}_{i,:})$
  $loss_i \leftarrow residuals_i$
  $MAT^{E}_{i,:} \leftarrow coeffs_i$
end for

4.1 Finding Similar Users

Once MAT^{E} is known we determine the similarity between any two users using methods that measure distances between vectors. Cosine and euclidean distances are the two methods we have used for determining similar users.

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:

$$cos\_dist(u_i, u_j) = 1 - \frac{(MAT^{E}_{i,:}).(MAT^{E}_{j,:})}{\|MAT^{E}_{i,:}\|\|MAT^{E}_{j,:}\|},$$

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

and,

$$euc\_dist(u_i, u_j) = \sqrt{\sum_{i=1}^{|T^{all}|}(MAT^{E}_{i,:} - MAT^{E}_{j,:})^2},$$

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

5 Experimental Results

The Model Context Protocol and our approach of using tool call count for personalization is new hence, existing personalization benchmarks like LaMP [13] cannot be used to validate the methods presented in this paper. We have generated new synthetic data in a SQLLite 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. The code to generate 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. 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.

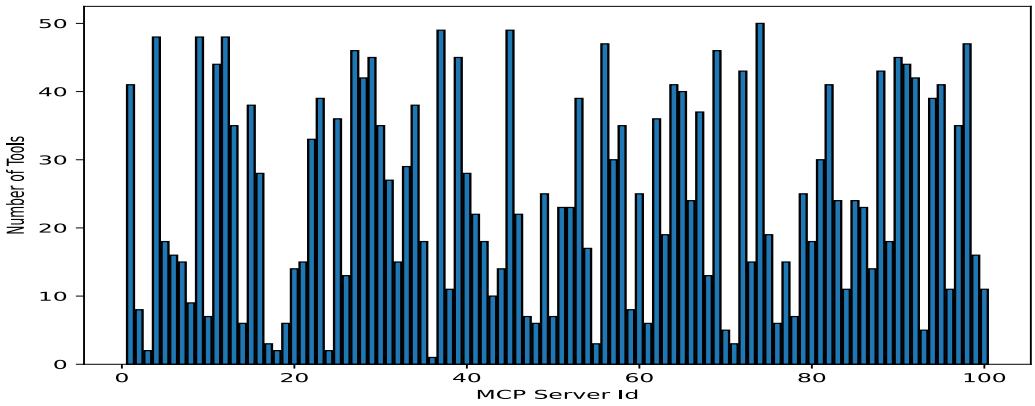
Results have been computed for embeddings dimensions of 8 and 32.

5.1 Canary Users

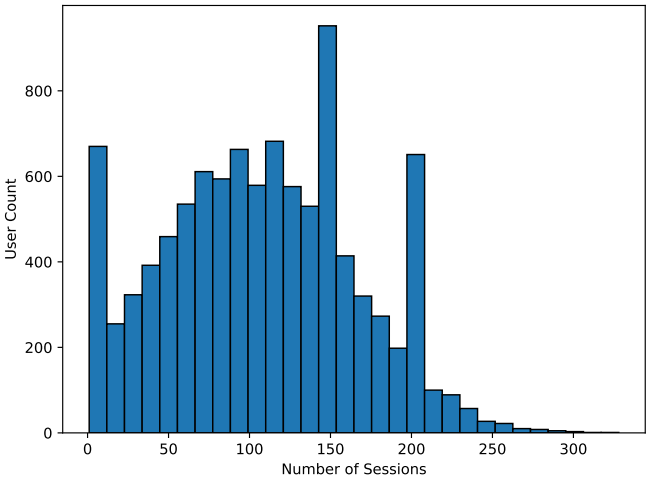
For a sample of users we have overwritten the randomly generated data so the sample has known attributes that help us determine the effectiveness of user embeddings.Data of 10% of randomly selected user's are modified to have very similar session and tool invocations. We classify these samples as canary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

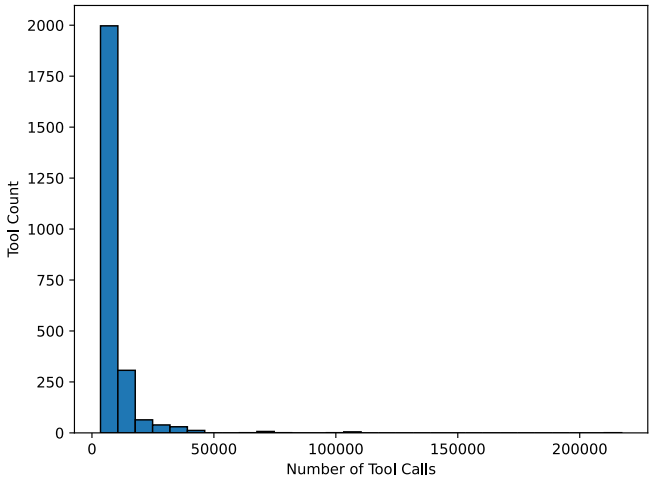
5.1.2 Canary Users 2 Composing the other 5% of the users, all users in this group have same number of sessions but the depth of randomly selected sessions has been decreased by 1. The cosine



(a) Number of tools in each MCP server



(b) Distribution: Number of sessions/user



(c) Distribution: Number of tool calls

Figure 1: Summary of synthetic data.

and euclidean distances in this group are expected to be close but not zero.

5.2 Results

In Figure 1, the main characteristics of the synthetic data are shown. These are number of tools in each MCP server and the distribution of sessions and tool call. From Fig. 1(c), it is observed that some tools are used more frequently than others, and this reflects practical scenarios where tools for common functions (e.g. summarization) will be used more often than others.

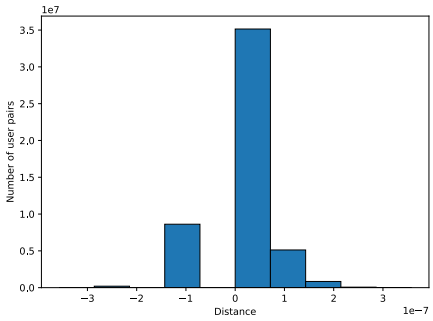
Results have been produced using Algorithms 2 and 3, and with different embedding dimensions of 8 and 32.

Figure 2 shows the results for Algorithm 2 with an embedding dimension of 8. Using Principal Component Analysis (PCA) we have also transformed the embeddings to 2 principal components, these are shown in figs. 2g - 2i. The principal components for Canary 1 users are exact which show up as a single point in Fig.2h. This is further corroborated as both the cosine and euclidean distances (2b and 2c) for all canary 1 users are in a single distribution.

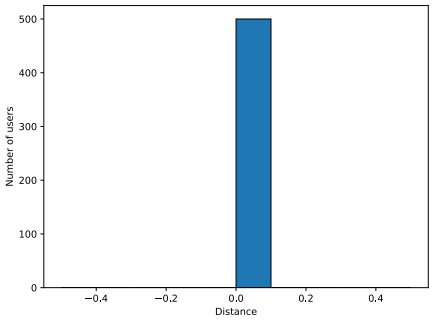
The euclidean distances for canary 2 (2f) show more variance than the cosine distance (2c) highlighting that euclidean distance is better able to distinguish the embeddings than cosine. Compared to the PCA of all users (2g), the PCA for Canary 2 users (2i) shows more closeness between embeddings and both the canary 1 and canary 2 results are on expected lines.

In personalization, it is helpful to know distinct clusters of users which can be tied to user attributes like persona, corporate roles etc. allowing an LLM to present unique experiences depending on the cluster the user belongs to. Using the elbow method (Within Cluster Sum of Squares - WCSS) we have determined an optimum number of clusters (one with lowest WCSS) (2j) and plotted them (2k), highlighting that embeddings can also determine the clusters to identify user attributes.

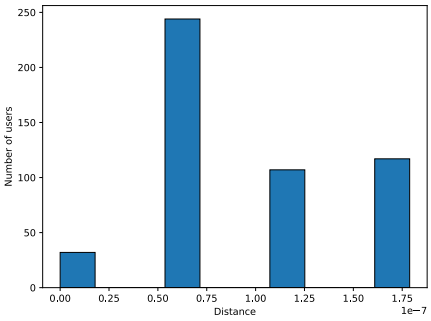
Figure 3, shows plots for algorithm 3 with embedding dimension of 8. Apart from converging much faster than algorithm 2,



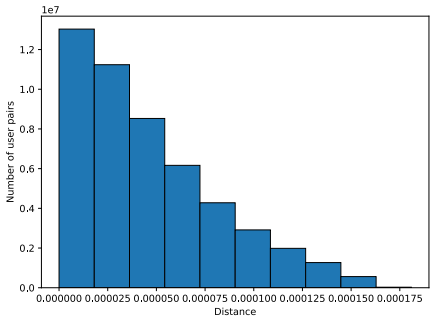
(a) Cosine distance between all users



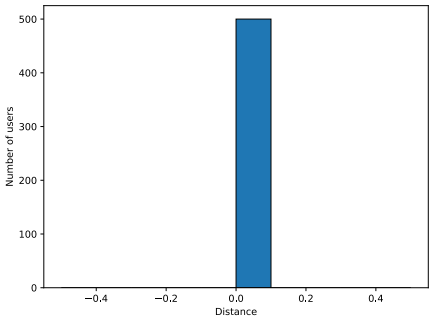
(b) Cosine distance between Canary 1 users



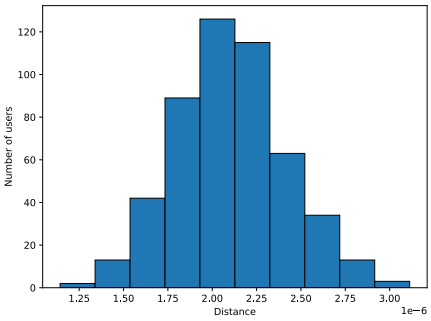
(c) Cosine distance between Canary 2 users



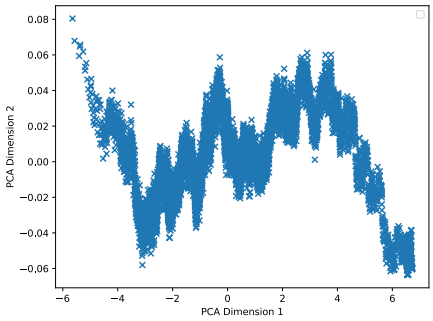
(d) Euclidean distance between all users



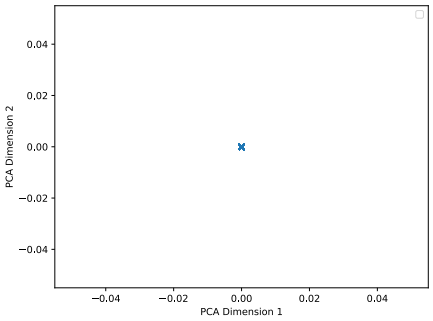
(e) Euclidean distance between Canary 1 users



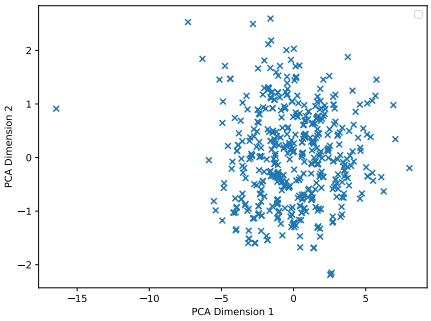
(f) Euclidean distance between Canary 2 users



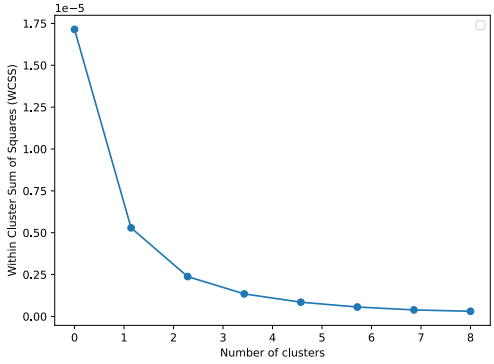
(g) Embeddings PCA for all users



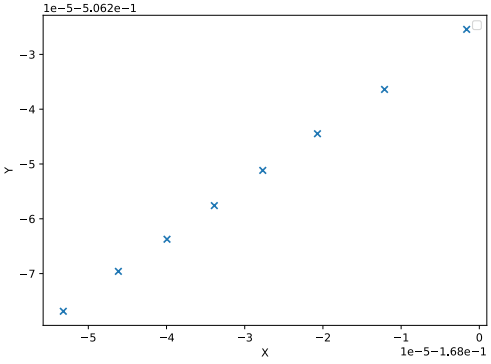
(h) Embeddings PCA for Canary 1 Users



(i) Embedding PCA for Canary 2 Users

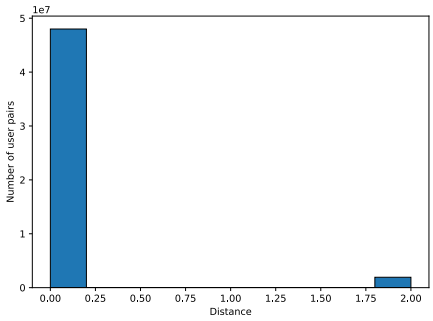


(j) Number of clusters using elbow method

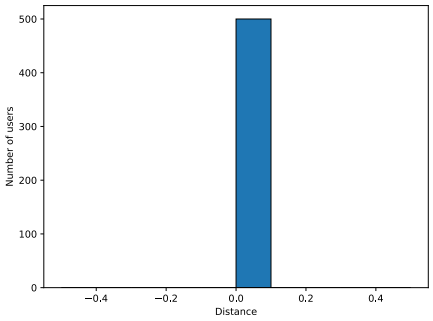


(k) Cluster Centroids

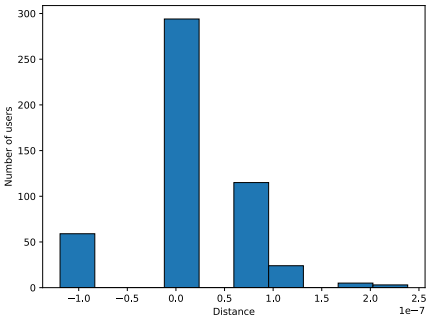
Figure 2: Embeddings-8: Results from Algorithm 2



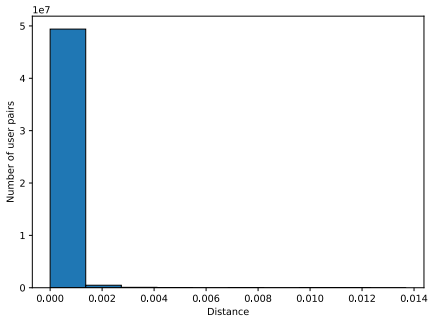
(a) Cosine distance between all users



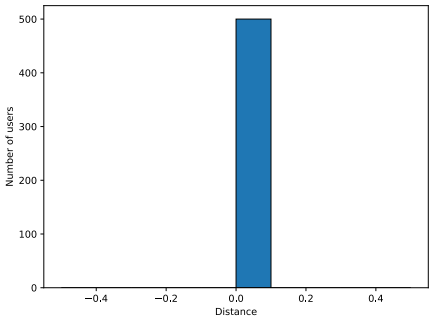
(b) Cosine distance between Canary 1 users



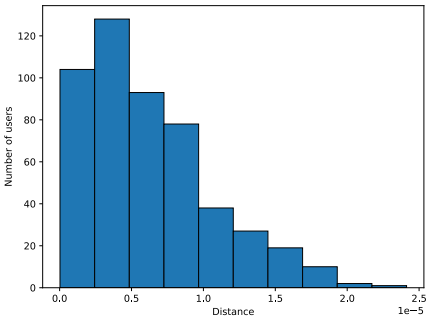
(c) Cosine distance between Canary 2 users



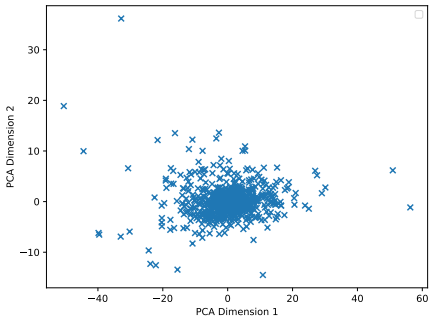
(d) Euclidean distance between all users



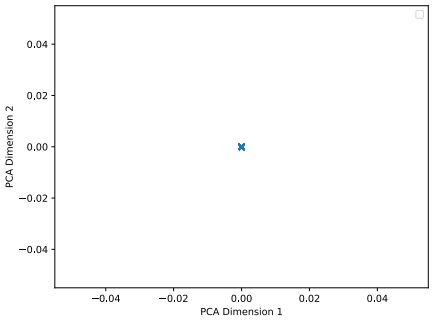
(e) Euclidean distance between Canary 1 users



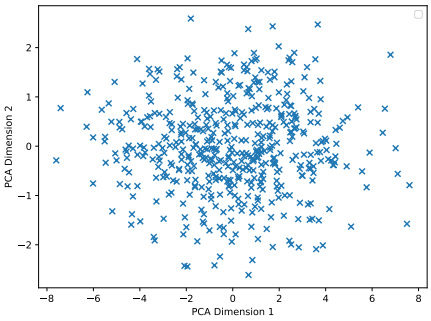
(f) Euclidean distance between Canary 2 users



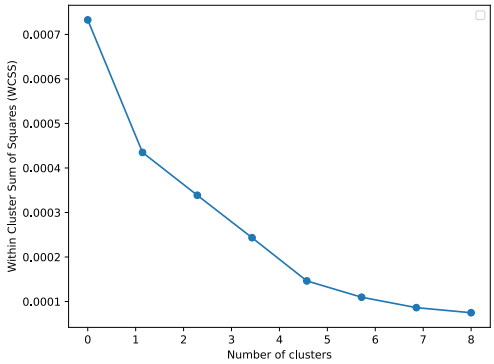
(g) Embeddings PCA for all users



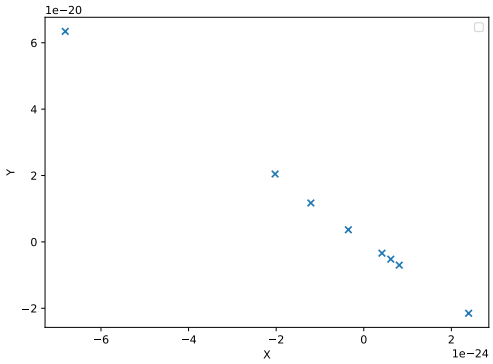
(h) Embeddings PCA for Canary 1 Users



(i) Embedding PCA for Canary 2 Users



(j) Number of clusters using elbow method



(k) Cluster Centroids

Figure 3: Embeddings-8: Results from Algorithm 3

the embeddings generated with algorithm 3 are more distinct as highlighted from the PCA for canary 2 (3i) and all users (3g). The cosine distances for canary 2 (3b) are also more spread out than with algorithm 2 (2b) further confirming this observation. While the number of clusters identified by alg. 2 (2k) is same as alg. 3 (3k) , the cluster co-ordinates are much different.

Figures 4 and 5 show similar data when the embedding dimension are higher at 32 (for brevity we have not plotted all data points, they are available in the extended version of this paper[]). Apart from more distinct buckets for cosine distance, we did not observe significant differences in distances and PCA between the embedding dimensions of 8 and 32, showing that a lower dimension embedding can be used. Though we feel that higher dimension embeddings will be more effective when user count is higher (than 10000) and determining this is slated for future work.

6 Conclusions and Future Work

In this paper, we have presented methods to generate low-dimension user embeddings from MCP interactions and utilizing synthetic data shown that using distance measures these embeddings can determine similar users as well as identify unique user clusters. The first algorithm 1 prepares and normalizes the data, following which we presented two different methods for embedding generation.

Our experiments on synthetic data show that the embeddings for all users which make 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 determined by PCA reduction to two dimensions. Using different sizes for the dense embeddings we showed that the smaller embedding size (8) can be used to distinguish the users, though further work is required to determine optimal embedding dimensions as number of user's increases.

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

6.1 Future Work

This work has set up a foundation for further work on generating embeddings based on MCP tools calls for the purpose of LLM personalization as a downstream activity. Future work will focus on the following areas:

- Our current algorithms assume a single tool call per prompt, but LLMs 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.
- While a larger embedding dimension will capture the characteristics of all users, they come with higher storage and computation costs. Determining relationship between number of users and an ideal embedding size remains an open area of research.
- 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 tools usage data is organically available.

References

[1] 2024. Model Context Protocol. https://modelcontextprotocol.io/. (Accessed: Sept. 22, 2025).

[2] 2026. Experiments Code Repository. https://github.com/cbelwal/Python/tree/main/MachineLearning/Research/UserEmbeddings_MCP/. (Accessed:Jan. 15, 2026).

[3] 2026. Synthetic Data Repository. https://github.com/cbelwal/Python/releases/download/UserEmbeddingsData_v1.0/mcp_interactions_u10000.zip. (Accessed:Jan. 15, 2026).

[4] Itsugun Cho, Dongyang Wang, Ryota Takahashi, and Hiroaki Saito. 2022. A Personalized Dialogue Generator with Implicit User Persona Detection. arXiv:2204.07372 [cs.CL] https://arxiv.org/abs/2204.07372

[5] 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. 2023. Large Language Models for User Interest Journeys. arXiv:2305.15498 [cs.CL] https://arxiv.org/abs/2305.15498

[6] Sumanth Doddapaneni, Krishna Sayana, Ambarish Jash, Sukhdeep Sodhi, and Dima Kuzmin. 2024. User Embedding Model for Personalized Language Prompting. arXiv:2401.04858 [cs.CL] https://arxiv.org/abs/2401.04858

[7] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. 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. https://openreview.net/forum?id=nZeVKeeFYf9

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

[9] Jiongnan Liu, Jiajie Jin, Zihan Wang, Jiehan Cheng, Zhicheng Dou, and Ji-Rong Wen. 2023. RETA-LLM: A Retrieval-Augmented Large Language Model Toolkit. arXiv:2306.05212 [cs.IR] https://arxiv.org/abs/2306.05212

[10] Jiongnan Liu, Yutao Zhu, Shuting Wang, Xiaochi Wei, Erxue Min, Yu Lu, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. 2024. LLMs + Persona-Plug = Personalized LLMs. arXiv:2409.11901 [cs.CL] https://arxiv.org/abs/2409.11901

[11] Lin Ning, Luyang Liu, Jiaxing Wu, Neo Wu, Devora Berlowitz, Sushant Prakash, Bradley Green, Shawn O'Banion, and Jun Xie. 2024. User-LLM: Efficient LLM Contextualization with User Embeddings. arXiv:2402.13598 [cs.CL] https://arxiv.org/abs/2402.13598

[12] Chris Richardson, Yao Zhang, Kellen Gillespie, Sudipta Kar, Arshdeep Singh, Zeynab Raeesy, Omar Zia Khan, and Abhinav Sethy. 2023. Integrating Summarization and Retrieval for Enhanced Personalization via Large Language Models. arXiv:2310.20081 [cs.CL] https://arxiv.org/abs/2310.20081

[13] Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2024. LaMP: When Large Language Models Meet Personalization. arXiv:2304.11406 [cs.CL] https://arxiv.org/abs/2304.11406

[14] Zhaoxuan Tan, Zheyuan Liu, and Meng Jiang. 2024. Personalized Pieces: Efficient Personalized Large Language Models through Collaborative Efforts. *CoRR* abs/2406.10471 (2024). https://doi.org/10.48550/ARXIV.2406.10471 arXiv:2406.10471

[15] Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. 2024. Democratizing Large Language Models via Personalized Parameter-Efficient Fine-tuning. *CoRR* abs/2402.04401 (2024). https://doi.org/10.48550/ARXIV.2402.04401 arXiv:2402.04401

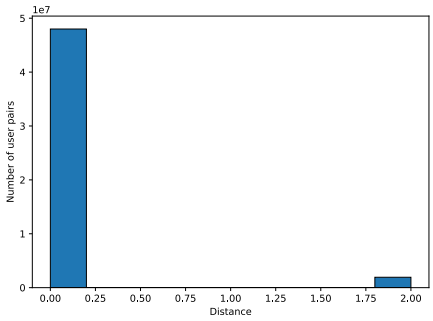
[16] Jaime Teevan, Susan T. Dumais, and Daniel J. Liebling. 2006. A study on the effects of personalization and task information on implicit feedback performance. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 449–458.

[17] 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. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). https://doi.org/10.48550/ARXIV.2302.13971 arXiv:2302.13971

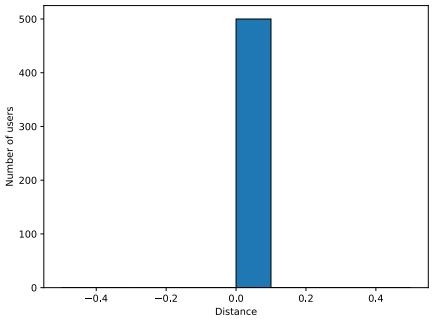
[18] Yu-Min Tseng, Yu-Chao Huang, Teng-Yun Hsiao, Wei-Lin Chen, Chao-Wei Huang, Yu Meng, and Yun-Nung Chen. 2024. Two Tales of Persona in LLMs: A Survey of Role-Playing and Personalization. arXiv:2406.01171 [cs.CL] https://arxiv.org/abs/2406.01171

[19] Bin Wu, Zhengyan Shi, Hossein A. Rahmani, Varsha Ramineni, and Emine Yilmaz. 2024. Understanding the Role of User Profile in the Personalization of Large Language Models. arXiv:2406.17803 [cs.CL] https://arxiv.org/abs/2406.17803

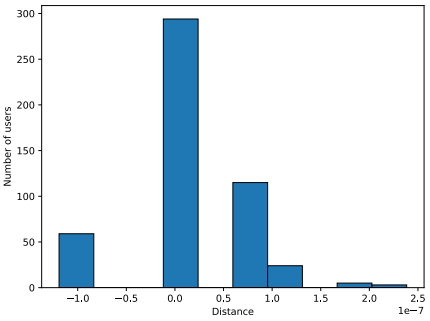
[20] Kai Zhang, Lizhi Qing, Yangyang Kang, and Xiaozhong Liu. 2024. Personalized LLM Response Generation with Parameterized Memory Injection. *arXiv preprint arXiv:2404.03565* (2024).



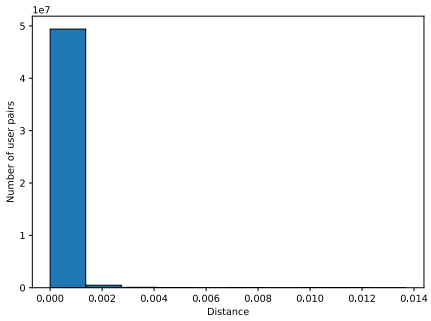
(a) Cosine distance between all users



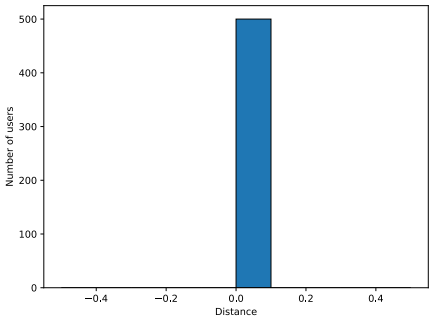
(b) Cosine distance between Canary 1 users



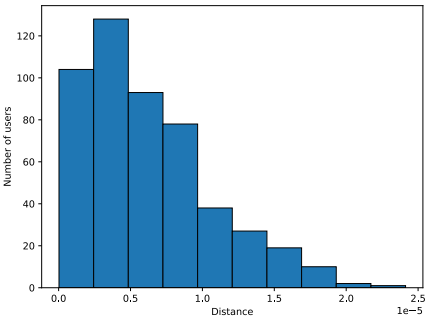
(c) Cosine distance between Canary 2 users



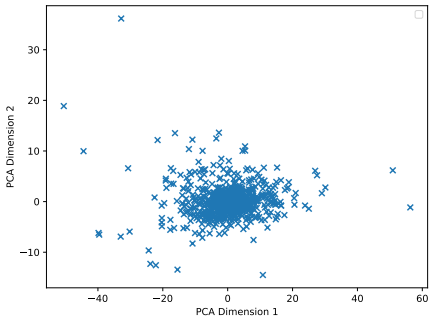
(d) Euclidean distance between all users



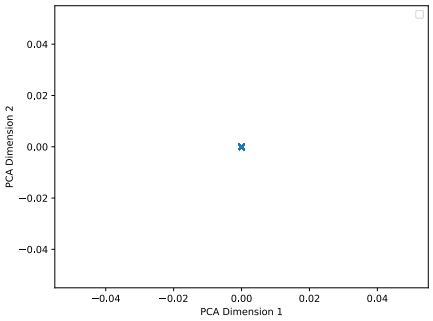
(e) Euclidean distance between Canary 1 users



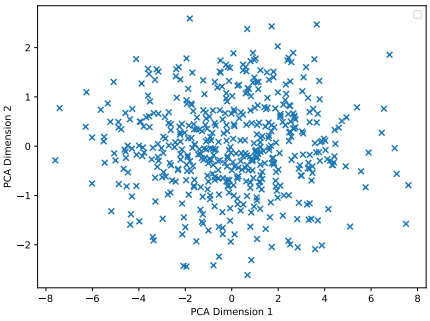
(f) Euclidean distance between Canary 2 users



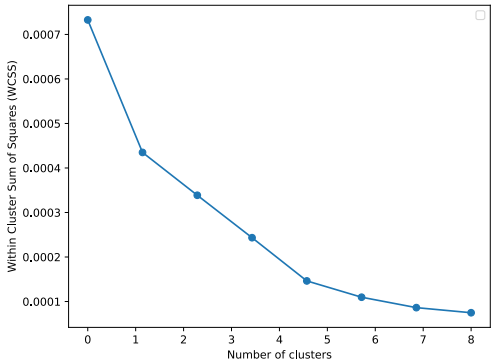
(g) Embeddings PCA for all users



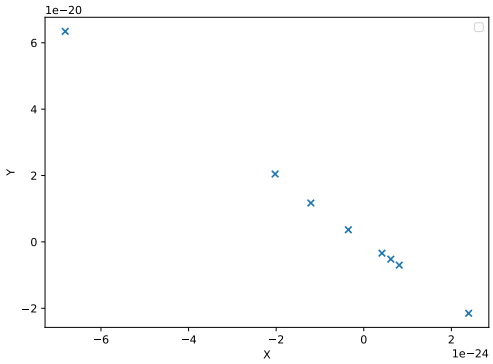
(h) Embeddings PCA for Canary 1 Users



(i) Embedding PCA for Canary 2 Users

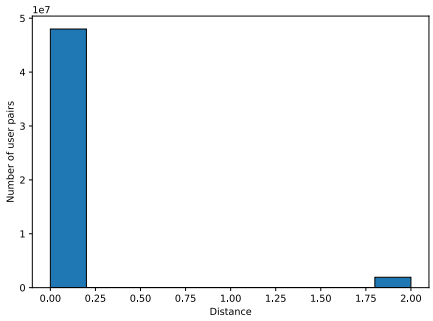


(j) Number of clusters using elbow method

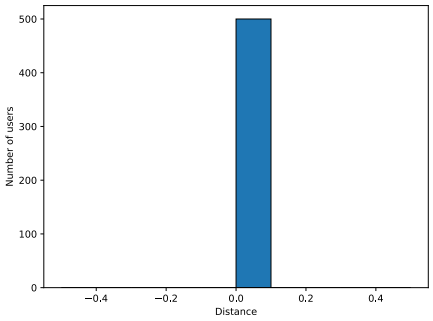


(k) Cluster Centroids

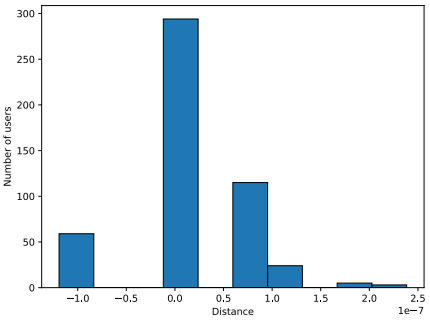
Figure 4: Embeddings-32: Results from Algorithm 2



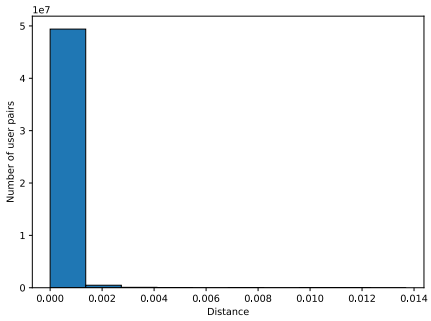
(a) Cosine distance between all users



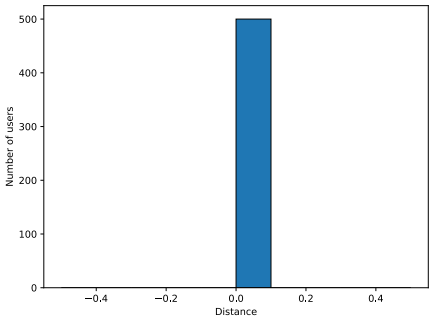
(b) Cosine distance between Canary 1 users



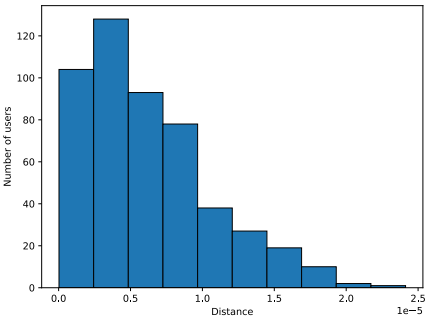
(c) Cosine distance between Canary 2 users



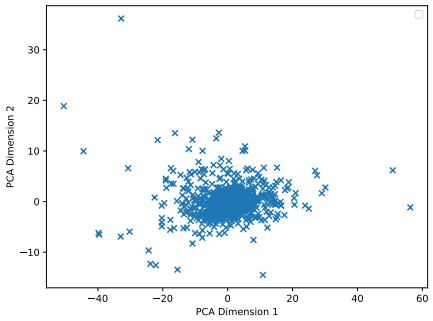
(d) Euclidean distance between all users



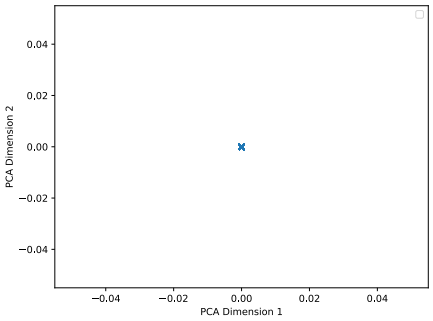
(e) Euclidean distance between Canary 1 users



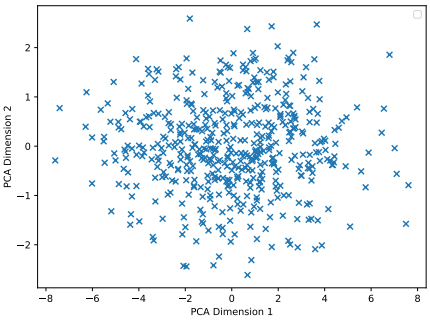
(f) Euclidean distance between Canary 2 users



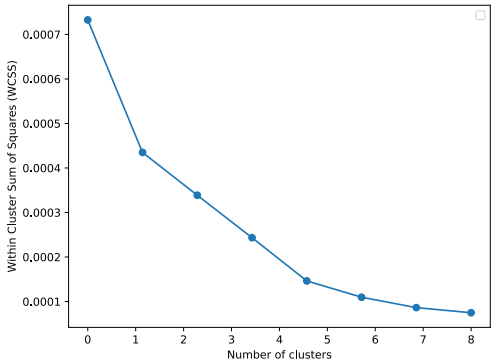
(g) Embeddings PCA for all users



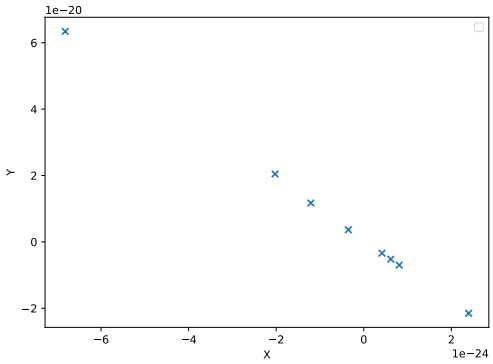
(h) Embeddings PCA for Canary 1 Users



(i) Embedding PCA for Canary 2 Users



(j) Number of clusters using elbow method



(k) Cluster Centroids

Figure 5: Embeddings-32: Results from Algorithm 3

[21] 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. 2025. Personalization of Large Language Models: A Survey. arXiv:2411.00027 [cs.CL] https://arxiv.org/abs/2411.00027

[22] Yuchen Zhuang, Haotian Sun, Yue Yu, Rushi Qiang, Qifan Wang, Chao Zhang, and Bo Dai. 2024. HYDRA: Model Factorization Framework for Black-Box LLM Personalization. arXiv:2406.02888 [cs.CL] https://arxiv.org/abs/2406.02888