XTrace WhitePaper

Abstract

XTrace is a decentralized protocol that provides AI agents with a secure, privacy-preserving memory and knowledge infrastructure. The protocol enhances efficient collaboration and autonomous learning while ensuring secure knowledge sharing among AI agents. By integrating cutting-edge AI and cryptography solutions, XTrace has developed an encrypted semantic search that operates at millisecond speeds. This functionality allows AI agents to perform retrieval-augmented generation on encrypted datasets using encrypted queries. The innovation establishes a secure, privacy-preserving knowledge base and enables the creation of shareable, portable, personalized memories for AI agents with modular access controls. Additionally, it introduces statefulness, empowering agents to recall and utilize knowledge from past interactions. These capabilities fundamentally transform how AI agents securely interact with datasets and other AI agents, unlocking a broad spectrum of applications.

Contents

1 Introduction

In today's data-driven landscape, businesses face the dual challenge of leveraging the vast potential of artificial intelligence while ensuring data security, privacy, and verifiability. As AI applications proliferate in both the B2B and B2C sectors, there is increasing demand for a robust solution that can ensure secure data access and foster continuous learning without compromising privacy.

XTrace meets these demands by establishing itself as the essential privacy-preserving data layer for AI agents. Our protocol is designed to be model-agnostic, enabling seamless integration across diverse AI ecosystems. This approach not only enhances data security, but also ensures that all AI agents, regardless of their underlying models, can operate within a framework that prioritizes data integrity and confidentiality.

At the core of XTrace technology is XRAG, a groundbreaking solution for semantic searches over encrypted text, capable of millisecond retrieval for each data chunk. This advanced capability allows for the creation of shareable, portable, and personalized memories and knowledge bases, all equipped with modular access controls for AI agents. Such customization and security mark the beginning of a new era of autonomy and statefulness for AI agents. XRAG enables these agents to learn from and build on previous interactions securely, ensuring that the confidentiality of the data is never compromised. This revolutionary capability lays the foundation for AI agents to operate with an unprecedented level of intelligence and discretion.

Building on the XRAG technology, XTrace equips its AI agents with personalized memory systems. These systems enable agents to retain valuable context from previous interactions and continuously adapt to user preferences and needs. As users maintain full control and ownership of their data, this secure, portable memory evolves with each interaction, making agents progressively more intelligent,efficient, and responsive. The ability to collaborate with various AI vendors while safeguarding data ownership enhances this adaptability. This continuous and dynamic learning loop offers businesses significant operational advantages, including resource optimization, cost reductions, and improved decision-making capabilities.

2 XRAG

XRAG is a cutting-edge framework that enables secure, privacy-preserving AI-driven data retrieval and collaboration across enterprises. At the heart of XRAG lies advanced cryptographic techniques, such as homomorphic encryption, Matryoshka Representation Learning, and quantized embeddings, allowing AI agents to query encrypted datasets without exposing the prompt. This approach ensures that AI agents can access relevant data securely while maintaining full end-to-end privacy for all parties involved.

One of the key innovations in XRAG is the use of homomorphic encryption to protect data during the query process. This ensures that even as AI agents interact with the data, no sensitive information is exposed or decrypted. XRAG's quantized embedding andreranking strategies further enhance the system's efficiency by performing only bit-level operations, eliminating the need for floating-point computations. This optimization drastically reduces the computational load while maintaining the integrity and privacy of the data.

2.1 RAG overview

Retrieval-Augmented Generation (RAG) is an AI technique that enhances large language models (LLMs) by integrating relevant information from external knowledge bases. Through semantic similarity calculations, RAG retrieves document chunks from a vector database, where these documents are stored as vector representations. This process reduces the generation of factually incorrect content, significantly improving the reliability of LLM outputs.[5]

A RAG system consists of two core components: the vector database and the retriever. The vector database holds document chunks in vector form, while the retriever calculates semantic similarity between these chunks and user queries. The more similar a chunk is to the query, the more relevant it is considered, and it is then included as context for the LLM. This setup allows RAG to

dynamically update an LLM's knowledge base without the need for retraining, effectively addressing knowledge gaps in the model's training data.

The RAG pipeline operates by augmenting a user's prompt with the most relevant retrieved text. The retriever fetches the necessary information from the vector database and injects it into the prompt, providing the LLM with additional context. This process not only enhances the accuracy and relevance of responses but also makes RAG a crucial technology in enabling AI agents to work with real-time data, making them more adaptable and effective in practical applications.

2.2 Embeddings and Similarity Measure

Queries to a vector database typically consist of unstructured data, which are converted into embeddings, represented as a vector $q \in R^n$. The data stored in a vector database can be represented as a matrix $D \in R^{m \times n}$. The retrieval process, given q, D and a similarity measure $f(u,v) : R^n \times R^n \rightarrow R$, is defined as:

$$\text{argmax}_i f(q, D_i)$$

In simpler terms, this process involves matching a query—whether in text, image, audio, or another format—with the most similar data in the database, based on the chosen similarity measure f. Typically, f might be cosine similarity, which captures semantic similarities between text embeddings. This allows the system to effectively match queries with the most relevant data stored, improving the accuracy and performance of information retrieval in various applications, including AI training and real-time data processing.

2.3 Matryoshka Representation Learning

The length of the embedding used in downstream tasks is a major factor when it comes to computational efficiency. In general, for retrieval, the shorter the length of the embedding, the faster the retrieval operation, but the less accurate the retrieved results. XTrace uses Matryoshka Representation Learning(MRL) to minimize the computational complexity of downstream retrieval tasks while retaining as much retrieval accuracy as possible.

MRL encodes information at different granularities and allows a single embdedding to adapt to the computational constraints of downstream tasks. Specifically, given any embedding model trained using training loss $L_r(x)$, an MRL loss is defined as

$$\mathcal{L}_{mrl}(x) = \frac{1}{\log N} \sum_{i=1}^{\log N} \mathcal{L}_r(x_{i:(i+1)\log N})$$

The MRL loss ensures that the information in the resulting representation is front loaded. That is, on a high level, the embeddings learned using MRL can be truncated while retaining as much information as possible. With MRL, XTrace is able to use binary embeddings of size 512 while retaining 90.76% retrieval accuracy, which hits a sweet spot when it comes to the speed and accuracy trade-off as shown in the metrics collected in table 1.

Dimension	1024	512	256	128	64
float32 Performance Retention (%)	100	95.22	86.01	67.34	34.25
binary Performance Retention (%)	96.46	90.76	79.52	60.31	32.28

Table 1: MRL Performance Retention vs Truncation + Quantization

2.4 Binary Embedding and Reranking

Since we are working with zero-knowledge (zk) and homomorphic encryption, we avoid performing calculations using floating points due to their inherent nondeterminism and high computational requirements. Instead, we propose using a quantization model that converts vectors into bit vectors with values 0 and 1 instead of floating-point embeddings. To quantize the floating-point embeddings into integer embeddings, we threshold the normalized embeddings at 0:

$$f(x) = \begin{cases} 0 & \text{if } x <= 0 \\ 1 & \text{if } x > 0 \end{cases}$$

In our system, we use trained binary embeddings for retrieval, with the training method detailed in the section on BERT-based two-stage retrieval. Our embeddings are represented as $q \in Z_2^n$ and the similarity measure to be Hamming Distance $H : Z_2^n \times Z_2^n \rightarrow N$:

$$H(u, v) = \sum_{i=1}^{n} u_i \oplus v_i$$

A smaller Hamming Distance indicates closer embeddings and, thus, higher document relevance. By utilizing Hamming Distance and binary embeddings, we enable partially homomorphic operations through bit manipulations, eliminating the computational overhead associated with floating-point operations in fully homomorphic encryption.

To maintain the performance of vector database retrieval, we implemented the re-ranking technique suggested by Yamada et al.[7] Initially, we retrieve rescore_multiplier × topk results using binary query and document embeddings, resulting in a list of the top k results from this double-binary retrieval. We then rescore this list of binary document embeddings using the float32 query embedding. The details of this process are introduced in the section on BERT-based two-stage retrieval. By doing so, we can still preserve 95% of the accuracy while working with bit data.

2.5 Binary Embedding as Locality Sensitive Hash

In this section we provide a mathematical proof that our binary embedding technique is a family of locality sensitive hash functions over R^d, and therefore the Hamming Distance computed over binary embedding preserves the relative distance measure in the vector space of the original real embeddings.

A locality sensitive hashing scheme is a distribution over a family of hash function F operating over a collection of objects, such that for every 2 objects x,y:

$$\mathbf{Pr}_{h \in F}[h(x) = h(y)] = \text{sim}(x, y)$$

where $\text{sim}(x,y) \in [0, 1]$. Such a scheme leads to a compact representation of objects so that similarity of objects can be estimated from their compact sketches, and also leads to efficient algorithms for approximate nearest neighbor search and clustering.[1]

Consider the following binary embedding scheme introduced in Charikar [1], given a collection of vectors in R^d, we define a family of hash function as follows: We randomly sample a vector r from a standard Gaussian distribution and define $h_r(x)$ as:

$$h_r(x) = I(r \cdot x \geqslant 0)$$

h_r is a family of locality sensitive hash function which follows from the fact that:

$$\mathbf{Pr}[h_r(x) = h_r(y)] = 1 - \frac{\theta(x, y)}{\pi} \approx 0.878 \cos\theta(x, y)$$

where $\theta(x,y)$ is the angle between vectors $x,y \in R^d$, and $\cos\theta(x,y)$ is a popular similarity measure used in information retrieval. This is proved in the work by Goemans and Williamso [4]

Now that we have established the fact that the binary embedding is in fact a locality sensitive hashing scheme, what remains to be shown is that the distance measure $1 - \text{sim}(x,y)$ in R^d is isometrically embeddable to Hamming Distance.

First notice that for any family of locality sensitive hash function h that corresponds to similarity measure $\text{sim}(x,y)$, there exists a *binary* locality sensitive hash function that corresponds to $\text{sim}'(x, y) = \frac{1+\text{sim}(x,y)}{2}$:

Consider a pairwise independent hash function $b(\cdot)$ that maps vectors to $\{0, 1\}$, it is easy to verify that $b(h(\cdot))$ is a family of locality sensitive hashing function, given that if $x = y$, $\mathbf{Pr}[b(x) =$

b(y)] = 1, and if x ≠ y: **Pr**[b(x) = b(y)] = 1/2:

$$\begin{aligned}\mathbf{Pr}[b(h(x)) = b(h(y))] &= \mathbf{Pr}[h(x) = h(y)] + \frac{1}{2}\mathbf{Pr}[h(x) \neq h(y)] \\ &= \mathrm{sim}(x,y) + \frac{1}{2}(1 - \mathrm{sim}(x,y)) \\ &= \frac{1 + \mathrm{sim}(x,y)}{2} = \mathrm{sim}'(x,y)\end{aligned}$$

Note that such a binary hash function family gives an embedding of objects into the Hamming cube (obtained by concatenating the values of all the hash functions in the family). For object x, let v(x) be the element in the Hamming cube x is mapped to. $1 - \mathrm{sim}'(x,y)$ is simply the fraction of bits that do not agree in v(x) and v(y), which is proportional to the Hamming distance between v(x) and v(y). Thus this embedding is an isometric embedding of the distance function $1 - \mathrm{sim}(x,y)$ in the Hamming cube. But

$$1 - \mathrm{sim}'(x,y) = 1 - (1 + \mathrm{sim}(x,y))/2 = (1 - \mathrm{sim}(x,y))/2$$

This implies that $1 - \mathrm{sim}(x,y)$ can be isometrically embedded in the Hamming cube.

2.6 Optimizations

In this section, we will present the optimization techniques we have designed to ensure our retrieval system achieves peak performance,scalability, and efficiency while maintaining privacy, verifiability, and security.

2.6.1 Indexing and Clustering

One common way to speedup the retrieval process of the vector database is to use clustering methods such as K-means to cluster similar data points together. When given a search query q, the database will compute, given set of cluster centroids C produced by K-means:

$$\mathrm{argmax}_{x \in \mathrm{argmax}_{c \in C} f(q,c)} f(q, x)$$

Note by doing clustering, instead of a scan over the entire database, we only need to iterate of a cluster identified by the most similar centroid compared to the search query, which is a significant performance boost. To determine the number of cluster k in our database we define the following measure a(i) for the data point i in cluster C_I, which captures the notion of how well i is assigned to its cluster:

$$a(i) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, i \neq j} f(i,j)$$

We then define the mean dissimilarity b(i) of point i to some cluster C_J as the mean of the distance from i to all points in C_J:

$$b(i) = \min_{J \neq I} \frac{1}{|C_J|} \sum_{j \in C_J} f(i,j)$$

Finally we can set the number of cluster k to be the value that maximize the following quantity, which by setup reflectshow well the data is clustered:

$$\frac{1}{N} \sum_{i=1}^{N} \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

2.6.2 BERT–based Two–Stage Retrieval

Retrieving documents using embeddings suffer from information loss from the process of turning documents into vector embeddings. To make retrieved documents more semantically similar to the user query, we can use re-ranking in a two-stage retrieval process. In first stage of the retrieval process, i.e. candidate generation stage, a regular indexed similarity search will be performed on the query vector and the document vectors stored in the database. Denote the set of document chunks returned in the first stage retrieval as D, a re-ranker R is a model that, given query q and first stage

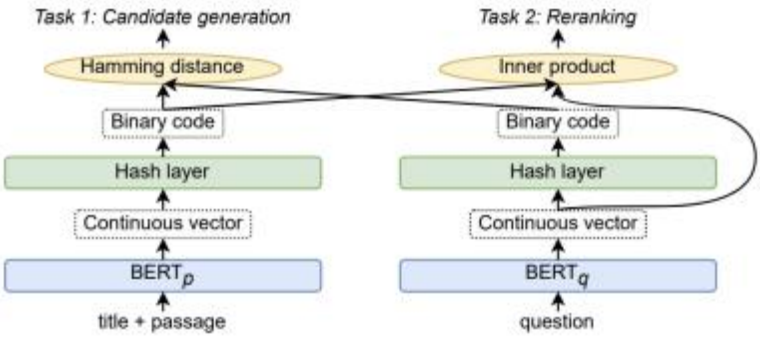

Figure 1: BERT-based Binary Passage Retriever

document set D, estimates the relevancy of q with respect to each document chunk $d_i \in D$. The process of using re-ranker to select the most relevant document chunks from D is the second stage retrieval,i.e. reranking stage. Specifically our two-stage BERT-based retrieval process contains the following:

- Binary Passage Retriever(BPR): BPR computes a binary embedding for each document chunk in the vector database by adding a hash layer on top of a BERT-encoder. Given embedding $e \in R^d$ outputed by BERT-encoder , the hash layer transform it into a binary code $h = \text{sign}(e)$
- Two-stage Retrieval: At the candidate generation stage, we efficiently obtain the top-l candidate document chunks using the Hamming distance between the binary code of query h_q and each passage in the vector database h_p. We then rerank the l candidate passages using the inner product between the continuous embedding of query e_q and h_p and select the top-k passages from thereranked candidates to return to the querier.
- Training BPR: Due to the addition of a hash layer, which uses sign function is not differentiable at 0, to train the BPR using gradient descent, we need to approximate the hash layer with a differentiable function. Specifically, during training, the hash layer is computed via:

$$\tilde{h} = \tanh \beta e$$

Where β is a scaling factor. It is easy to verify that $\tilde{h}$ converges to h as $\beta \to \infty$. Given training data $\{q_i, p_i^+, p_{i,1}^-, \ldots, p_{i,n}^-\}_{i=1}^m$, the retriever is trained using the following loss: $L = L_{\text{cand}} + L_{\text{rerank}}$, where

$$\mathcal{L}_{\text{cand}} = \sum_{j=1}^{n} \max(0, -(\langle \tilde{h}_{q_i}, \tilde{h}_{p_i^+} \rangle + \langle \tilde{h}_{q_i}, \tilde{h}_{p_i^-} \rangle) + \alpha)$$

$$\mathcal{L}_{\text{rerank}} = -\log \frac{\exp(\langle e_{q_i}, \tilde{h}_{p_i^+} \rangle)}{\exp(\langle e_{q_i}, \tilde{h}_{p_i^+} \rangle) + \sum_{j=1}^{n} \exp(\langle e_{q_i}, \tilde{h}_{p_i^-} \rangle)}$$

Evidence in Yamada, et al. [7] has shown that the two-stage retrieval process using BERT-based binary retriever outperforms the single-stage retrieval process in multiple information retrieval benchmark datasets.

2.7 Privacy Preserving XRAG

This section outlines the protocol by which an AI agent securely retrieves contextual data from its encrypted knowledge base using XRAG technology. The process begins when a user inputs a prompt, which the AI agent converts into a bit vector. This vector is then encrypted homomorphically. The encrypted prompt is transmitted to the XRAG network, where homomorphic operations are executed on the ciphertext to isolate the most pertinent data chunks. The top-k ciphertext data chunks

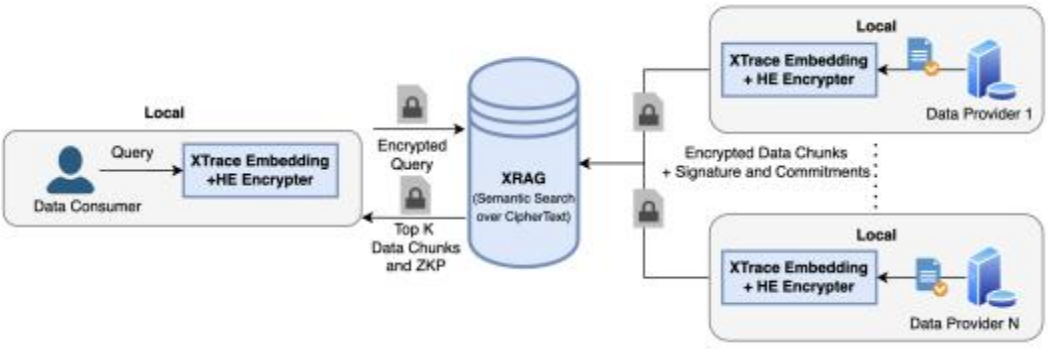

Figure 2: XRAG Overview

are subsequently returned to the data consumer, ensuring that the entire search process maintains strict security protocols.

The protocol described here is designed for both data providers and queriers within the XTrace system. While it currently utilizes the Paillier encryption scheme, it is also compatible with other homomorphic encryption schemes, allowing for adaptability and broader application.

Data Preprocessing

1. **Setup**: Data Provider conducts the following step locally:
   (a) Paillier Key Generation $\mathsf{Gen}() \rightarrow (\mathsf{sk}_{\mathsf{hom}}, \mathsf{pk}_{\mathsf{hom}})$
   (b) KZG Setup $\mathsf{Setup}(\lambda, F) \rightarrow \mathsf{gp}$
   (c) BLS Signature $\mathsf{Gen}() \rightarrow (\mathsf{sk}_{\mathsf{Bls}}, \mathsf{pk}_{\mathsf{Bls}})$
2. **File upload**: The data provider preprocesses the file by splitting it into chunks, running binary embeddings on each chunk, and then executing the following encryption scheme for each chunk's embedding m_i :
   (a) Encrypt each chunk with $\mathsf{Enc}(\mathsf{pk}_{\mathsf{hom}}, m_i) \rightarrow \mathsf{ct}_i$
   (b) KZG Commit to each data chunk $\mathsf{Com}(\mathsf{gp}, m_i) \rightarrow \mathsf{comm}_i$
   (c) BLS Sign each data chunk with $\mathsf{Sig}(\mathsf{sk}, \mathsf{comm}_f) \rightarrow \sigma_i$
   (d) Upload $\mathsf{ct}_i, \mathsf{comm}_i, \sigma_i$ for each data chunk m_i to XRAG database

Query

1. **Local Prompt Preprocessing:**
   (a) Run binary embeddings on the given query q
   (b) Specify the data provider to work with and obtain $\mathsf{pk}_{\mathsf{hom}}, \mathsf{gp}, \mathsf{pk}_{\mathsf{Bls}}$
   (c) Encrypt the prompt with $\mathsf{pk}_{\mathsf{hom}}$ and send it to XTrace database
   (d) Generate key pair $\mathsf{pk}_{\mathsf{querier}}, \mathsf{sk}_{\mathsf{querier}}$ to get encrypted data and deposit tokens on Smart Contract
2. **Semantic Score Calculation:**
   (a) Perform homomorphic operation to calculate the hamming distance between the ciphertext and the encrypted query Hamming Distance$(\mathsf{Enc}(\mathsf{embed}_{\mathsf{query}}), \mathsf{Enc}(\mathsf{embed}_{\mathsf{chunk}})) \rightarrow \mathsf{res}$
   (b) Return the encrypted semantic results res to the data provider
   (c) Data providers decrypts the result with $\mathsf{sk}_{\mathsf{hom}}$ and perform re-ranking on top of the similarities
   (d) Send back data chunks with highest similarity and encrypt it with the public key from the querier $\mathsf{Enc}(\mathsf{pk}_{\mathsf{querier}}, m_i) \rightarrow \mathsf{ct}_{\mathsf{res}}$ and the KZG evaluation $\mathsf{Eval}(\mathsf{gp}, f, u) \rightarrow \pi$ back to the querier

2.8 Benchmark TFHE vs Paillier vs XTrace Optimized Paillier Retrieval

We have implemented and benchmarked the performance of our system with both TFHE scheme implemented by Zama [2] and Paillier scheme implemented by Intel.[3] In addition, we have also conducted an XTrace Optimized Paillier Encryption Solution. The following is the performance we get by running the system on a Amazon M6i large EC2 with Intel Xeon 8375C (Ice Lake) CPU, Ubuntu 24.04 OS and 8GB RAM. All documents are represented as binary embeddings of length 512.

Crypto System	Preprocessing&KeyGen	Encrypt Query	Apply Calculation	Decrypt Result
Cosine Similarity Plaintext	0	0	72ms	0
TFHE	862ms (per query)	71ms	2212ms	0.5ms
Paillier	256ms (one time setup)	659ms	36ms	291ms
Paillier XTrace Optimized	256ms (one time setup)	5.6ms	0.094ms	2.8ms

Table 2: Time Per Query Per Chunk

Crypto System	Circuit Size	Cipher Size	PK/SK Size	Evaluation Key Size
TFHE	5KB	16.7MB	8KB	211.4MB
Paillier XTrace Optimized	N/A	0.001511 MB	2KB	N/A

Table 3: Memory Footprint

3 Memory Sharing Layer for Agents

In the rapidly advancing field of artificial intelligence, the development of a memory layer for AI agents represents a critical innovation, enabling these agents to not only perform tasks but also to remember and learn from each interaction. This memory layer, essentially a dynamic and evolving database, stores experiences and data in a way that AI agents can access and utilize in future tasks, thereby mimicking the human ability to build on past knowledge. Such a system is pivotal in transforming AI agents from mere computational tools to intelligent entities capable of contextual understanding and improved decision-making. By continuously integrating new data while maintaining rigorous security and privacy standards, the memory layer ensures that AI agents evolve into more sophisticated and personalized assistants. In this section, we'll explain the reason why Agents need memory and provide the XTrace's memory design for AI agent.

3.1 Why Agents Need Memory Layer

Large language models (LLMs) have a notable limitation due to context length constraints, which cap the amount of information they can retain and reference at any given moment. This limit means that LLMs struggle with understanding or recalling long conversations, documents, or complex tasks that extend beyond their maximum token capacity, typically a few thousand tokens. As interactions with LLMs become longer or involve more intricate subject matter, the models often "forget" earlier parts of the conversation or document, leading to repetitive responses, loss of context, or degraded performance. This constraint highlights the need for effective memory solutions in LLMs, enabling them to retain, retrieve, and apply information across interactions without reloading the same details repeatedly. A memory component would allow LLMs to maintain continuity in conversations, handle multi-step tasks,and draw on prior interactions, making them far more efficient and effective in complex applications.

While state-of-the-art language models have achieved remarkable improvements in extending context length, they still face significant challenges when it comes to reasoning over long, complex inputs. Long context lengths theoretically allow these models to "see" more information at once, which can be helpful in scenarios like summarization or document review. However, simply increasing the amount of accessible text does not inherently enhance a model's reasoning abilities. These models often struggle with synthesizing and

integrating information meaningfully over extended contexts, especially when complex relationships or logical steps are required to generate accurate responses[6]. Without the capability to truly understand and reason across long-form text, these LLMs tend to fallback on shallow pattern matching rather than deep inference. This limitation calls for more robust memory architectures and improved attention mechanisms, enabling models not only to access vast amounts of information but also to reason effectively across it, retaining essential context and drawing conclusions with greater coherence and accuracy.

3.2 Memory Hierarchy

XTrace builds a two-tier memory system for all agents connected to XTrace's knowledge network. A XTrace managed universal memory that contains "common knowledge" which all agents have access to, and a personalized memory for each individual agents which is secured with XRAG and access control. This structure allows agents to store individual-specific knowledge while ensuring privacy and secure data sharing. By combining universal and personalized memory layers, XTrace enables agents to balance shared insights and individualized learning, which supports more consistent, contextualized, and privacy-conscious AI operations across the network.

3.2.1 XTrace Universal Memory

The universal memory layer in XTrace serves as a shared knowledge repository that all connected agents can access, providing a foundational layer of "common knowledge." This layer includes information and insights that are broadly useful across various contexts, which could encompass standardized data, public knowledge, industry-specific guidelines, or other reference material that is essential for agents working on diverse but interrelated tasks. By storing and managing this shared knowledge centrally, XTrace's universal memory allows agents to instantly reference important information without the need to independently process or store it themselves, reducing redundancy and saving processing power.

The universal memory layer also fosters collaboration among agents by acting as a single source of truth. Agents can align their decisions and responses based on consistent, verified knowledge,which minimizes discrepancies that might arise from using isolated datasets or individual interpretations. This uniformity is crucial when agents need to work together across complex or multi-step tasks, as it ensures that each agent operates with an identical understanding of core principles and facts. Additionally, the universal memory layer can be regularly updated with the latest, verified information, providing all agents with real-time access to up-to-date data.

By offloading common knowledge to the universal memory, XTrace allows individual agents to focus their personalized memory on context-specific details, user preferences, or unique task insights, all of which remain private and secure. This layered memory approach not only enhances efficiency but also enables agents to work together more effectively, accessing shared knowledge as a baseline while still preserving the personalized learning and nuanced memory required for specific tasks and user interactions.

3.2.2 Personalized Memory

As the proliferation of AI agents continues, we envision a future where every human will have a personalized AI agent, which remembers all the contexts of their past interactions and takes actions. Such an AI agents will require a auxiliary memory system, which XTrace provides.

The personalized memory layer in XTrace is designed to store and manage information unique to each agent, enabling them to retain specific details and learnings relevant to their individual tasks and interactions. Unlike the universal memory, which holds commonly accessible knowledge, personalized memory captures the unique context, preferences, and past interactions pertinent to each agent, creating a more tailored experience and allowing agents to build continuity over time. This layer empowers agents to remember prior interactions with specific users or projects, draw on historical data relevant to particular tasks, and maintain nuanced understanding without re-processing foundational details every time.

XRAG ensures that personalized memory is accessible only to the designated agent and within the scope of authorized queries, preserving user privacy through stringent access

control and cryptographic techniques. This privacy-focused approach means that each agent's personalized memory is isolated and encrypted, accessible only when relevant and without exposing sensitive data across the network. This setup not only preserves data confidentiality but also aligns with principles of data ownership and user consent, allowing users or organizations to maintain control over their private information.

3.3 Working Memory Implementation: Paging with LRU Eviction

To address the challenge of limited context windows in large language models, XTrace has introduced a Memory Paging system with LRU (Least Recently Used) Eviction, inspired by the virtual memory architecture commonly used in operating systems. This approach allows agents to manage memory more dynamically, effectively creating a "working memory" that feels nearly limitless despite the constraints of finite context space.

Memory Paging works by dividing the agent's memory into "pages" or manageable data chunks. As the agent interacts and processes tasks, only the most relevant pages are loaded into the context window, with less immediate data paged out. When the agent's context window reaches its limit, the LRU Eviction strategy prioritizes the removal of the least recently used pages, making space for new, relevant information. This method allows agents to seamlessly retrieve past context as needed while reducing the processing load and preventing memory from becoming cluttered with outdated or irrelevant information.

This system provides agents with virtually unlimited memory space and maintains low performance overhead, as the paging mechanism only retrieves what is contextually important at each moment. It enables agents to remember long-term interactions, past decisions, and task histories with continuity, without compromising efficiency or performance.

References

[1] Moses S. Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the Thiry–Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 380–388. ISBN: 1581134959. DOI: 10.1145/509907.509965. URL: https://doi.org/10.1145/509907.509965.

[2] Ilaria Chillotti et al. *TFHE: Fast Fully Homomorphic Encryption over the Torus*. Cryptology ePrint Archive, Paper 2018/421. 2018. URL: https://eprint.iacr.org/2018/421.

[3] Intel Corp. *Intel Paillier Cryptosystem Library: An ISO–compliant Implementation Accelerated with Intel AVX512/IFMA*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/homomorphic-encryption/iso-compliant-paillier-cryptosystem-library.html.

[4] Michel X. Goemans and David P. Williamson. "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming". In: *J. ACM* 42.6 (Nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: https://doi.org/10.1145/227683.227684.

[5] Patrick Lewis et al. *Retrieval–Augmented Generation for Knowledge–Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[6] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[7] Ikuya Yamada, Akari Asai, and Hannaneh Hajishirzi. *Efficient Passage Retrieval with Hashing for Open–domain Question Answering*. 2021. arXiv: 2106.00882 [cs.CL]. URL: https://arxiv.org/abs/2106.00882.