SocialGenPod: Privacy-Friendly Generative AI Social Web Applications with Decentralised Personal Data Stores

Vidminas Vizgirda
s1750767@ed.ac.uk
University of Edinburgh
Edinburgh, UK

Rui Zhao
rui.zhao@cs.ox.ac.uk
University of Oxford
Oxford, UK

Naman Goel
naman.goel@cs.ox.ac.uk
University of Oxford
Oxford, UK

ABSTRACT

We present SocialGenPod, a decentralised and privacy-friendly way of deploying generative AI Web applications. Unlike centralised Web and data architectures that keep user data tied to application and service providers, we show how one can use Solid — a decentralised Web specification — to decouple user data from generative AI applications. We demonstrate SocialGenPod using a prototype that allows users to converse with different Large Language Models, optionally leveraging Retrieval Augmented Generation to generate answers grounded in private documents stored in any Solid Pod that the user is allowed to access, directly or indirectly. SocialGenPod makes use of Solid access control mechanisms to give users full control of determining who has access to data stored in their Pods. SocialGenPod keeps all user data (chat history, app configuration, personal documents, etc) securely in the user's personal Pod; separate from specific model or application providers. Besides better privacy controls, this approach also enables portability across different services and applications. Finally, we discuss challenges, posed by the large compute requirements of state-of-the-art models, that future research in this area should address. Our prototype is open-source and available at: https://github.com/Vidminas/socialgenpod/.

CCS CONCEPTS

• **Information systems** → **Collaborative and social computing systems and tools**; *Data management systems.*

KEYWORDS

Retrieval Augmented Generation; Decentralised Web; Privacy; Solid

1 INTRODUCTION

Many Web applications using generative AI like large language models (LLMs) [11] and diffusion models [10] have emerged recently. Applications like ChatGPT, DALL-E, Perplexity, Maket.ai, Synthesia, WriteSonic, Jasper are plentiful in domains such as
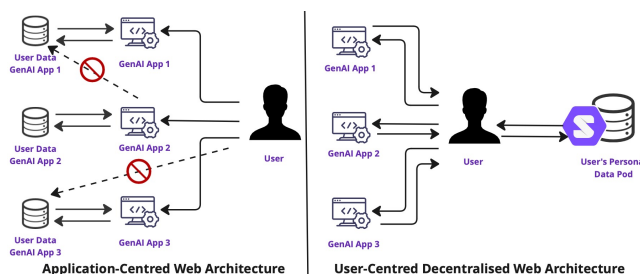
Figure 1: Centralised architecture (left) makes it difficult for users to have control over their data and use their data across applications. Decentralised architecture (right) decouples data from applications, giving users control over their data and making it easier for them to use their data across apps.

search, writing, music, and design. Such applications usually manage the data of their users in a centralised way. For example, in a chat application, data from different users (such as user queries and answers generated by an LLM) are managed by the chat application provider, usually using a centralised database. Users have little control over how this personal data is used by the application provider [6]. Centralised storage of sensitive user data also raises other privacy concerns. For example, in March 2023, a technical glitch in ChatGPT allowed some users to see parts of other users' conversations [3].

Other than control and privacy concerns, another drawback of centralised and application-specific data management is that users cannot easily re-use their data from one application in another (also known as the vendor lock-in problem). For example, a user may not be able to use their data easily from an application for conducting background research in another application for writing or preparing graphics. Ideally, people should be able to use their data in the best applications available in the market, without unnecessary obstructions. This is illustrated in Figure 1.

To address these issues, we demonstrate a decentralised and privacy-friendly way of deploying generative AI Web applications. The key idea in this work is to use decentralised personal data stores for managing user data as illustrated on the right of Figure 1. Specifically, we use Solid [7, 8], a decentralised Web specification based on standard, open, and interoperable data formats and protocols. It allows users to store their data in their Web-accessible Pods (Personal Online Data Stores) and configure granular access control for applications and other Pod/WebID owners. Solid also handles secure transmission of data for authorised requests. To support data portability, Solid apps and services read and write data to the users'

Pods instead of siloed app backends. Users can transfer their data from one Pod service provider to another, or even host one themselves. But there are also several technical challenges in using Solid with generative AI models. One challenge is the large computation power required for running these models, which makes an entirely decentralised and privacy guaranteeing implementation difficult. While user queries and generated responses can be stored in a user's Pod, large AI model inference cannot run locally (for example, in browser) with reasonable performance on typical end-user devices. Another challenge for RAG (Retrieval Augmented Generation) — a technique which uses personal documents stored in user Pods for grounding the responses — is that similarity search is required to pick out relevant documents. Existing Solid server implementations do not offer any support for similarity search, neither with vector embedding techniques nor other frequently used approaches, like Okapi BM25.

In the paper, we discuss how we overcome some of these challenges for building our prototype and future research directions that will be of interest to the attendees of TheWebConf. Attendees will also be able to interact with the prototype. Interested community members can also contribute to further development of the open-source prototype.

2 RELATED WORK

There are several related approaches for decentralised and privacy-friendly deployment of generative AI models. These include training highly capable small models (e.g., models with fewer parameters, distilled models, quantised models, etc) that can run with less compute and making the models openly available. Examples include the LLaMA models from Meta [9], MoE models from Mistral AI [5], and many others available on the open-source hub HuggingFace.

Several emerging applications rely on local generative AI models to limit data privacy concerns with retrieval augmented generation. For example, Rewind gathers sensitive user data, such as screen recordings, emails, and meeting summaries, to enable searching past user activity. Their approach is to minimise data that leaves the user's computer by using local device resources for AI model inference [1]. Similar ideas are also used by projects such as WebLLM, PrivateGPT and ChatDocs. However, running models locally is often infeasible on end-user devices due to the high compute requirements of large AI model inference. Local models are also not sufficient in use cases that require data sharing between users.

We also note the "AI gateway" approach, such as the one implemented by llm-gateway. AI gateways act as a proxy between users and hosted models, intercepting user data before it is sent to external models, and scrubbing personal information or performing other security checks. This approach also avoids vendor lock-in by providing a unified API for interacting with different services. While there is an overlap in the motivation, the AI gateway approach focuses on single-user single-application functionalities, without features for collaboration between applications and users.

Tim Berners-Lee recently published a design note [2]. There are some similarities in the ideas proposed in his design note and SocialGenPod. But, to the best of our knowledge, SocialGenPod is the first concrete implementation of RAG using Solid.

3 OVERVIEW OF SocialGenPod

Example Scenario: Let us consider an example use-case scenario (Figure 2) that requires Retrieval Augmented Generation (RAG) and user data sharing. Suppose there are two friends or collaborators: Alice and Bob. Alice stores personal documents, such as her notes on a particular project, in her Solid Pod. She then configures a virtual "personal AI assistant" to have read access to this data. The virtual personal assistant is a Web app with Alice's configuration and is powered by an embeddings model for retrieval and a Large Language Model (LLM). The models may be running either on Alice's machine or on an external service. Bob can chat with this virtual assistant (using the Web app) and learn from Alice's project notes (potentially avoiding scheduling an unnecessary meeting), without ever getting a copy of the full documents. Alice could also configure access permissions so only a specific set of friends or users could interact with her personal AI assistant. Figure 3 shows SocialGenPod prototype using RAG with private data.

In the above example, the defining feature of Alice's personal AI assistant is Alice's configuration of the chatbot Web app, i.e., which service providers it uses and where it sources Alice's documents from. The AI models themselves are not necessarily personalised — it is access to Alice's documents that makes it a personal assistant. The Web app itself could run anywhere (for example, Alice could host it or share the source code for others to run or point to a trusted public app provider, run by, e.g., a government or union).
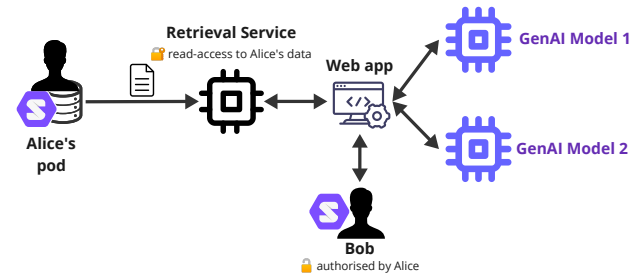

Figure 2: Example Web application data flow in SocialGenPod. Alice allows Bob to interact with or query her personal data through a Web app. The app uses a retrieval service to find permitted documents in Alice's Solid Pod that are most relevant to Bob's query. The app then uses one of the available generative AI models to generate a contextualised response for Bob.

The above use-case example can be generalised to other domains where private data sharing is relevant, for example, commercial consulting, financial advice, medical advice, career advice, meeting scheduling, or personalised feedback. It does not have to be restricted to just language generation either — the same scenario could apply to multi-modal models too. The data stored in Solid Pods can also include also other kinds of data such as chat history between two users, chat history between a user and an LLM, etc. Solid Pods store all these data in a privacy-friendly and user-centric manner, and subject to user specified preference, can be used by various innovative and useful generative AI applications (personal AI assistants being one example).

(a) Without RAG

(b) With RAG

Figure 3: RAG with private data in SocialGenPod. A chatbot without RAG (top) is unable to answer a user's query. The same chatbot with RAG (bottom) produces a better response.

4 DESIGN AND USAGE DETAILS

SocialGenPod is built using Solid, leveraging its mechanisms for access control, user authentication, and data discovery. As shown in Figure 2, SocialGenPod contains several modular components: a Web application, a retrieval service, and one or more Generative Artificial Intelligence (GenAI) models. These components are all generic and substitutable. We provide demonstrable implementations of all these components in an end-to-end prototype. In practice, Web app developers and service providers would develop their own respective Apps or services that integrate easily with other components in the ecosystem.

In SocialGenPod, the user interacts with a Web app, which establishes the user's identity using Solid-OIDC. The app retrieves and stores user data such as configuration and chat history in their Solid Pod. The user may manually configure the app to select their preferred providers for retrieval and generative AI models.

The retrieval and generative AI services may run externally. The generative AI services do not have direct access to the Web app user's Pod. This allows the user to easily switch between different service providers. We expect these services to be similar to Solid Pod services – a marketplace of public or private, free or paid, fast or slower but high-accuracy services for the user to choose from (or self-host), depending on the quality of service, reputation, and trust.

On the other hand, based on the use case, the retrieval service will require access to the Pod it sources data from (which may or may not be the Web app user's Pod). This is subject to user-specified preferences such as which services can be used to access their data, which data, and for what purpose.

Restricted (e.g., private or paid subscription) model services can authenticate users using Solid-OIDC, which eliminates the need for self-developed account management mechanisms. For example, the model service can maintain an internal list of allowed users or read lists of trusted users from Solid Pods. It allows features like paid subscription and delegation of usage rights to friends.

While service providers could be deployed on any suitable infrastructure, it would be sensible for Solid Pod providers to offer such services alongside Pod hosting. As Solid Pod providers are already trusted by their users to store their data securely, this would eliminate the need to trust another third party. It would also boost performance, by potentially allowing data to be transferred between services running on the same host with low latency.

Although external services may cache user data for performance, this should be ephemeral and securely isolated. Since the service providers are expected to not store user data permanently, it is easier to establish trust and identify dishonest behaviours, thus encouraging trustworthy services.
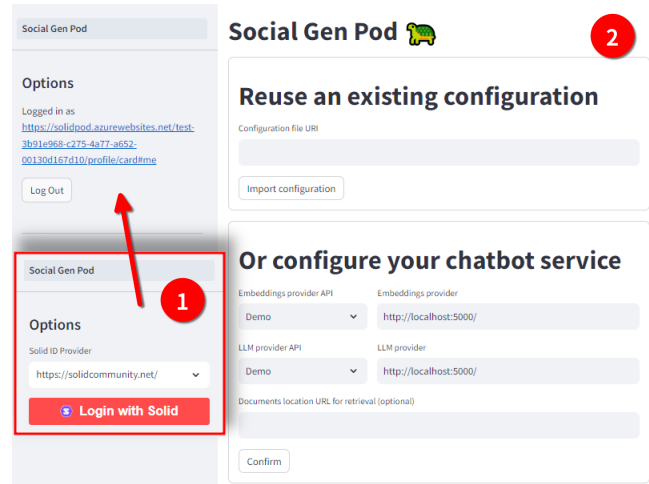
4.1 The Web App

Figure 4: Login and configuration menu in SocialGenPod.

Our SocialGenPod implementation uses an embeddings model provider as the retrieval service and a Large Language Model provider as the generative AI model service.

Upon opening the SocialGenPod Web app, the user logs in using their Solid credentials (Step 1 in Figure 4). Then, they can set up the service providers to use (or reuse a previous configuration) for retrieval and generation (Step 2 in Figure 4). Once the app is configured, as shown in the left sidebar in Figure 5, the user can choose to start a new conversation thread or continue an existing conversation. Chat threads and messages are stored in the user's Pod. Available models from selected providers are also shown in the sidebar, so the user can quickly switch between different models at any time, even mid-conversation.

The Web app handles data retrieval and updates from/to the user's Pod, and the language model service only receives conversation data from the Web app. When sending requests to external
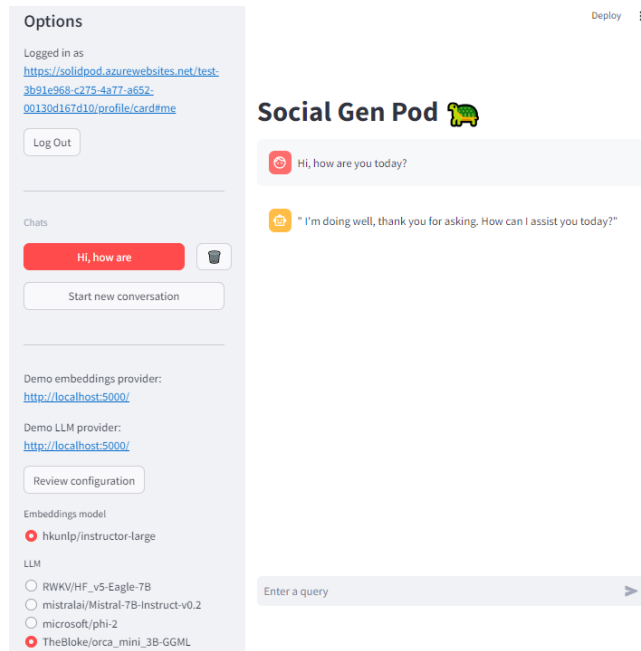
Vidminas Vizgirda, Rui Zhao, and Naman Goel

Figure 5: Full chat interface in SocialGenPod, showing chat threads and available models.

endpoints, the Web app sends Solid-OIDC authentication headers as well, so both the Solid Pod and the external services can confirm the identity of the requestor. In this way, the services can perform *user* (but not *account*) management, such as only allowing a predefined set of users to use the service.

Optionally, in the configuration step, the user may provide a Solid Pod location to source documents from for Retrieval Augmented Generation. If provided, the specified retrieval service gets relevant documents, which are sent together with user queries to the language model service. An alternative design choice could have been for the Web App to retrieve documents and send them to the retrieval service for similarity search, however, our chosen approach supports advanced social sharing, as discussed in subsection 4.2.

4.2 Social Sharing

Besides individual use cases, SocialGenPod supports social sharing of both service providers and personal documents.

The retrieval service can use Solid-OIDC for user identity verification. This way, Alice can selectively allow only Bob to read data from her Pod *using a particular retrieval service*, independently of which front-end Web app Bob uses. Since the generative AI model provider is decoupled from the retrieval service provider, Bob could use any generative AI model in this workflow, self-hosted or remote.

The advantage of this micro-services-oriented plugin architecture is that it keeps services minimal and substitutable. Service providers do not need to maintain account management systems nor associated controlled data storage. As briefly discussed above, this reduces the complexity and vulnerability of these services, and could improve trust between service providers and end users.

5 LIMITATIONS AND FUTURE WORK

With SocialGenPod, we demonstrated how one can build privacy-friendly generative AI Web applications using Solid. In future work, it would be useful add more personalisation features, such as fine-tuning on personal data and deploying personalised models in a decentralised way. The main challenges to address are delegating fine-tuning computation and inference on personalised models.

Another potential direction for future research could be addressing challenges related to document retrieval by relevance. Currently, SocialGenPod relies on a retrieval service provider copying personal documents for local computations, which requires trusting that the provider will handle data securely.

It would also be interesting to explore private inference techniques [4], for improved privacy protection when using untrusted model providers. Other privacy and security risks related to personal AI assistants and methods to mitigate those risks should also be investigated in future work. For example, a malicious actor with access to a personal AI assistant could potentially attempt to retrieve the full contents of user's personal documents by prompting the assistant many times.

ACKNOWLEDGMENTS

REFERENCES

[1] Brett Bejcek. 2024. Who has access to my data? https://help.rewind.ai/en/articles/6526621-who-has-access-to-my-data. Accessed: 2024-02-05.

[2] Tim Berners-Lee. 2023. Inference from Private Data. https://www.w3.org/DesignIssues/PrivateData.html. Accessed:2024-02-05.

[3] Ben Derico. 2023. ChatGPT bug leaked users' conversation histories. https://www.bbc.co.uk/news/technology-65047304. Accessed: 2024-01-24.

[4] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. 2022. Iron: Private inference on transformers. *Advances in Neural Information Processing Systems* 35 (2022), 15718–15731.

[5] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[6] Emanuele La Malfa, Aleksandar Petrov, Simon Frieder, Christoph Weinhuber, Ryan Burnell, Raza Nazar, Anthony G Cohn, Nigel Shadbolt, and Michael Wooldridge. 2023. Language Models as a Service: Overview of a New Paradigm and its Challenges. *arXiv e-prints* (2023), arXiv–2309.

[7] Essam Mansour, Andrei Vlad Sambra, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. A demonstration of the solid platform for social web applications. In *Proceedings of the 25th international conference companion on world wide web*. 223–226.

[8] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. 2016. Solid: a platform for decentralized social applications based on linked data. *MIT CSAIL & Qatar Computing Research Institute, Tech. Rep.* (2016).

[9] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[10] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. 2023. Diffusion models: A comprehensive survey of methods and applications. *Comput. Surveys* 56, 4 (2023), 1–39.

[11] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).