ChatGPT

Efficient Personalization of LLMs via Compact Context Representations

Personalizing a large language model (LLM) for millions of users requires injecting user-specific context efficiently, without adding long text prompts or huge per-user caches. Recent research and industry techniques focus on **compressing extensive user context (e.g. thousands of tokens) into brief continuous representations** – such as learned prompt vectors or latent memory slots – that steer the model. These methods aim to preserve personalization quality while **minimizing inference overhead**, making them suitable for ~4B-parameter class models in production. Below, we review key approaches for compact context injection and compare their expressiveness, storage needs, training effort, and runtime performance.

Soft Prompt and Prefix Tuning Techniques

Prompt tuning and **prefix tuning** replace long textual prompts with learned continuous vectors that serve as a "soft prompt." Instead of prepending a 5000-token user profile, a small set of trained embeddings is prepended to the input sequence or inserted into the model's layers [1] [2]. Li and Liang (2021) showed that *prefix tuning* (learning synthetic key/value vectors per Transformer layer) can steer a frozen model as effectively as fine-tuning, with far fewer parameters [3] [4]. Similarly, Lester et al. (2021) found that *prompt tuning* (optimizing a short sequence of input embeddings) can match full fine-tuning performance at scale, using only on the order of 10^4 learnable parameters [2]. Deep or "multi-layer" prompt variants (e.g. P-Tuning v2) inject learned vectors at multiple layers, further improving influence without modifying the backbone weights.

For personalization, one naive approach is to train a unique soft prompt for each user. In small experiments this can capture user-specific style or preferences, confirming that even a *single learned token* can impart a surprising amount of conditioning [5]. However, training and storing a separate prompt per user (millions of prompts) is impractical. Recent solutions therefore **generate prompts on the fly from compact user data**, rather than storing static prompts for each user. This retains the efficiency benefits – *soft prompts add minimal token overhead and hardly affect inference latency* [6] – while scaling to large user bases.

Encoder-Driven Context Compression (User Embeddings to Prompts)

A prominent strategy is to **compress a user's history into a small embedding or prefix** via a lightweight encoder, and inject that into the LLM. This pipeline typically involves a *user-context encoder* that produces a dense vector (or a few vectors) from the user's data, followed by a mapping that turns this vector into a set of "soft tokens" consumed by the LLM [7] [8]. The LLM itself remains frozen or only lightly fine-tuned. Such methods replace long textual profiles with *brief continuous prompts* that capture the same information density [9] [10].

Embedding-to-Prefix (E2P): One example is E2P [11], which maps a pre-computed **user embedding** directly into the LLM's hidden space as a single soft prefix token [12] [6]. A simple two-layer MLP "prefix projector" generates a personalized token embedding from the user's vector, which is then prepended to the input sequence [13] [14]. This approach personalizes the model **without any change to the LLM's weights**, avoiding costly fine-tuning [6]. Despite using just one learned token per user, E2P achieves significant gains on dialogue personalization (Persona-Chat) and recommendation tasks by leveraging the dense info in user vectors [15] [16]. Notably, in a production setting for music recommendations, E2P could take a 120-dimensional behavioral embedding (updated from a streaming service's user data) and inject it as a prefix to generate tailored playlist responses [17]. This yields personalization with essentially no token overhead (one extra token) and very low latency, since the user's context is encoded in a fixed small vector rather than thousands of tokens [6].

PERSOMA: Extending this idea, PERSOMA (Personalized Soft Prompt Adapter) uses a more elaborate two-stage encoder. It first **encodes the user's interaction history** (potentially many events) into a set of embeddings, then **resamples and compresses** these into a handful of prompt vectors via a learned adapter network [7] [8]. For example, a user's long history of movie ratings or dialogue turns can be distilled into, say, *N* soft prompt tokens. These tokens are then prepended alongside the task prompt to a frozen LLM (e.g. PaLM 2) to generate a personalized output [18]. PERSOMA explored different adapter architectures (MLP vs. small Transformer or Perceiver) and found that even a simple MLP adapter can effectively handle large histories [19] [20]. In experiments on a MovieLens genre prediction task, an MLP-based soft prompt adapter significantly outperformed both raw text prompts and static user embeddings, achieving higher F1 scores on personalization [21] [20]. This demonstrates that *compressing extensive user histories into soft prompt embeddings can retain fine-grained personalization signals* while keeping inference efficient. The adapter approach is compatible with parameter-efficient tuning techniques (like LoRA) for training – e.g. PERSOMA applied LoRA on the adapter and still achieved near full fine-tuning performance [22].

Persona-Plug: Liu et al. (2024) propose a similar plug-and-play personalization module called *Persona-Plug (PPlug)*. It employs a *lightweight user embedder network* to encode all of a user's historical context into a single user-specific embedding [23]. This embedding is then **attached to the LLM's input** (for instance, by concatenating it with the task prompt or as special prefix tokens) to condition generation [24]. The base LLM does not need to be fine-tuned at all – only the small embedder is trained on user data. Compared to retrieving a few raw history snippets as in-context examples, this learned embedding captures the user's overall style and preferences more coherently [24]. On the LaMP benchmark for language model personalization, PPlug's method of modeling the entire user history into one vector yielded significantly better personalized outputs than retrieval-based prompting [23].

Key advantages: Encoder-to-prompt pipelines **achieve high-quality personalization with minimal runtime cost**. They leverage powerful pre-trained LLMs (frozen) and feed them a dense representation of user specifics. Because the injected prompt is short (often just 1–5 tokens), it adds almost no latency or token length burden during inference [6]. These prompts can be generated quickly from stored user embeddings or on-the-fly via small networks. The storage footprint is also small – e.g. one 512-D vector per user (a few kilobytes) instead of thousands of tokens or an entire fine-tuned model per user. Training is efficient, usually requiring only PEFT on the projector or adapter module (a tiny fraction of the LLM's parameters) [5] [6]. Importantly, this approach scales to *millions of users*: each new user just gets an embedding, without retraining the whole LLM or extending its context window. It also avoids "prompt injection" issues of having to concatenate long profile text (which can hit context limits or be interpreted

incorrectly). The main trade-off is that the **expressiveness is capped by the dimensionality and quality of the learned embedding** – but experiments show that a well-trained compact embedding can encode rich personal traits, often yielding outputs on par with using the full history in text [25] [15].

Cross-Attention Latent Memory Integration

An alternative to feeding personal context as additional tokens is to modify the model architecture to *ingest external context via cross-attention*. In the style of DeepMind's **Flamingo** model, one can insert dedicated cross-attention layers that allow the LLM to attend to separate latent representations (e.g. a set of user memory vectors) in parallel to its normal self-attention on the input text [26] [27]. Flamingo was originally designed for vision-language tasks: it interleaves new gated cross-attention blocks into a frozen language model so that image embeddings (as keys/values) can influence text generation [26]. The same architectural idea can be applied to personalization, by treating a user's profile or preferences as an **external memory** that the model can query with cross-attention at each generation step.

In a memory-augmented LLM, the base model runs as usual, but at each decoder layer it has an extra attention head attending over a set of *memory key-value pairs* (which encode user-specific info) [27]. Those memory vectors might come from an encoder or a database of user information. Crucially, they do **not need to be appended to the text input**, so they don't consume context tokens. The transformer's self-attention remains focused on the actual prompt and generated tokens, while cross-attention separately fuses in the user context [28] [27]. This decoupling can be more compute-efficient for large memories: e.g., attending to 50 memory vectors of dimension *d* has cost O(50·N·d) for sequence length *N*, instead of O(N^2) if those were added as a long prefix to the sequence. It also lets the model **dynamically query parts of the user context as needed**, rather than passively reading a fixed prompt.

Research on retrieval-augmented transformers demonstrates the benefit of such mechanisms. For instance, some long-term dialogue architectures integrate a vector memory of past interactions and use a cross-attention interface to incorporate retrieved memory entries at each decoder block [29] [27]. This *Flamingo-style* design was found to improve coherence and personalization in dialogues with manageable latency overhead [30] [28]. Compared to a single-shot compressed prompt, a latent memory allows **larger or more fine-grained user context** to be available (since it can hold multiple vectors or slots) and potentially be updated continuously. In practice, implementing this requires adding new layers or heads to the model and fine-tuning them on personalized data. The upside is a potentially richer conditioning signal – the model can "decide" which memory aspects to focus on via attention – and reduced reliance on extremely high-density single-vector encodings.

However, architectural approaches come with engineering complexity. One must train the cross-attention layers (as done in Flamingo, where the LM was gated and fine-tuned to incorporate images [26]). The base model is no longer entirely frozen, though the modifications can be parameter-efficient if the new layers are small. In production, this approach is less common so far, but it points toward **LLMs that have a built-in interface for external personalization memory** instead of treating all context as plain text. For example, one could imagine a 4B decoder model with a small adapter that cross-attends to a user profile vector – essentially learning to treat that vector as latent context. This would avoid having to prepend any tokens at all to the user's prompt. Early results suggest that cross-attention integration can outperform naive prefix-based fusion in multimodal settings [31] [32], and it may similarly yield higher-quality personalization if properly trained. The trade-off is that it *increases model complexity and inference cost per token slightly*, as

each decoding step involves attending to extra keys/values (albeit a fixed small number). Thus, organizations will weigh this against simpler prompt-injection methods.

Retrieval-Augmented Personalization Hybrids

To handle extremely large or evolving user contexts efficiently, many systems turn to **retrieval** combined with projection. Rather than compressing an entire user history into one vector, the idea is to **store many pieces of user data externally and retrieve only the most relevant pieces for a given query**. Those retrieved pieces (which could be past interactions, preferences, or mini profiles) are then injected in compact form. This hybrid approach ensures that at inference time the model only sees a *focused, minimal context* representing the user, avoiding both long prompts and irrelevant data.

One simple form is **profile memory lookup**: maintain a database of vectorized user interactions (e.g. each past conversation turn or item liked is embedded as a vector). When the user makes a new query, the system finds the top-k relevant memory vectors (using similarity search) and then projects those into the model input. The projection could be as straightforward as concatenating the *k* retrieved items as a short prompt (if they are already short), or encoding them into a few latent tokens via an adapter. Crucially, this limits the token count – for example, retrieving 3 key facts about the user and providing them as a 3-sentence prefix, instead of dumping the entire history. Some memory-augmented LLM frameworks explicitly do this: they **retrieve top vectors from a user's memory store and feed them through either prefix tokens or a cross-attention mechanism** in the decoder [29] [27].

Another line of work tries to achieve personalization by **combining small learned components** retrieved from a pool. *Personalized Pieces (PER-PCS)* [33] is a recent framework where users "share" fragments of fine-tuned adapters. In training, a subset of users fine-tune PEFT modules (like LoRA or adapter parameters) on their data; these personalized adapters are then split into smaller *pieces* and added to a global library [33]. At inference, a new target user can retrieve a few relevant adapter pieces from the library (e.g. from users with similar profiles) and **assemble a custom adapter on the fly** [33] [34]. By gating and weighting these pieces, the model personalizes its behavior without training a full adapter for the new user. PER-PCS demonstrated performance *on par with one-person-one-model fine-tuning* but at a fraction of the storage and computation cost [35]. Essentially, it trades off some personalization fidelity for huge gains in scalability: many users can be served by mixing and matching a limited set of learned "persona pieces." This approach avoids per-user KV caching or large prompts; instead, it yields a tiny adapter (a few thousand parameters) per user assembled from reusable parts.

In practice, retrieval-based personalization offers a nice **flexibility and scalability**. Systems like recommendation chatbots can maintain an external vector index of user facts and preferences. As the user's behavior evolves, new vectors are added and old ones pruned – without touching the core LLM. When generating a response, the latest relevant vectors are fetched and injected via a compact prompt or prefix. This keeps the context fresh and targeted, and the inference load constant (since only a fixed number of vectors are ever injected). The main overhead is the retrieval step, but approximate nearest neighbor search is fast even over millions of entries [36] [37]. The **expressiveness** of this approach is high: in principle the model has access to a wide range of personal facts, but filtered to what likely matters for the query. The **storage footprint** lives in the external memory (which can be large, but is often an on-disk or distributed store, not loaded into the model's weights or context window). Training effort may be lower as well – one can use pre-trained encoders for the memory and need only ensure the LLM is capable of consuming retrieved info (possibly via slight fine-tuning or prompt engineering). Many current *RAG*

(Retrieval-Augmented Generation) systems for personalization use this template: e.g. retrieve a summary of the user's recent actions and a snippet of their profile, then feed those as a short prompt to the LLM. The hybrid of retrieval + compression thus helps **avoid both long prompts and per-user model bloat** by intelligently reusing information.

Comparing Approaches and Trade-offs

Each of the above techniques strikes a different balance between personalization fidelity, efficiency, and complexity:

- **Soft Prompt Tuning (Static per User):** *Expressiveness:* Moderate – a prompt of a few tokens can encode user preferences but might miss nuances if too short. *Storage:* Requires storing a custom prompt (continuous vectors) for every user; a single 5–10 token prompt per user is only a few KB, but millions of users mean a large total (though still far less than full models per user). *Training & Integration:* Easy to integrate at inference (just concatenate the learned prompt tokens) [6], and training uses standard PEFT on a small parameter set [2]. However, training one prompt per user doesn't scale well unless done in a multi-task fashion. *Runtime Performance:* Excellent – adds only a handful of tokens to the input, so negligible impact on latency or GPU memory. No dynamic retrieval needed; the personalization is instant via the prompt vectors.

- **Encoder/Adapter-Based Compression:** *Expressiveness:* High – by encoding potentially thousands of tokens of history into a rich vector or set of vectors, these methods preserve fine-grained personal context (e.g. overall style, long-term preferences) [24] [25]. *Storage:* Very efficient – store either a single user embedding (often hundred dimensions [17]) or nothing at all if the history can be encoded on the fly. One small encoder model is shared across all users. *Ease of Training:* Only the encoder/adapter (which could be an MLP, small transformer, etc.) needs training or fine-tuning [18]. This can be done on aggregated data for many users, making it feasible to cover millions. The main LLM stays fixed or only lightly tuned. *Inference:* The extra cost is computing the user representation (which can sometimes be precomputed). Injecting the resulting soft prompt is fast – often just 1–5 added tokens – so prefill and generation latencies remain low [6]. This makes it practical for real-time use, as evidenced by deployments in recommendation systems and personalized assistants.

- **Cross-Attention Memory Integration:** *Expressiveness:* Very high – the model can attend to a *bank of personal vectors* or memory slots, allowing it to draw on more information than a single-vector prompt could hold. It can also condition on the query to focus on relevant parts of memory at each step. *Storage:* External memory can be large (potentially many vectors per user), but the model itself doesn't have to carry those permanently. Each user's memory could be kept in a database or generated as needed. *Complexity:* This approach requires architectural changes (inserting cross-attn layers or similar) and thus more involved **fine-tuning and integration effort** [26] [27]. Maintaining and updating the memory module also adds system complexity (one needs to decide how to encode new user events into memory, how to prune it, etc. [29] [36]). *Runtime:* If the memory set per user is kept fairly small (dozens of vectors), cross-attention adds only a linear cost per decoding step, which is manageable. There may be a slight increase in time-to-first-token since the model processes extra keys/values, but because these memory vectors don't contribute to quadratic self-attention, the overhead is much less than including equivalent context as input tokens. This method shines when long-term personalization must be tightly integrated into generation (e.g. an AI companion

remembering many past dialogues), but it may be overkill for simpler personalization and is less widely adopted due to implementation challenges.

- **Retrieval + Projection Hybrids:** *Expressiveness:* High and *adaptive* – by fetching different pieces for different queries, the model isn't limited to a one-size-fits-all summary. It can access niche or rarely used user info on demand. For example, if a user suddenly asks about a years-old preference, a retrieval step can surface that detail which a fixed embedding might have "forgotten." *Storage:* All user data is stored externally (e.g. vector indexes, knowledge bases). This can scale horizontally (common in industry: e.g. a search index for user logs). The model itself remains small, with maybe a shared encoder for vectors. *Training:* Requires tuning the retrieval pipeline (to ensure relevant bits are selected) and possibly training the LLM to utilize retrieved info correctly (some methods fine-tune the LLM with retrieved context so it learns to interpret those vectors or texts). But each user doesn't need individual training. *Runtime:* Inference involves an extra step: e.g. nearest-neighbor search in a vector store. This can add a few milliseconds to each request, but modern ANN search is efficient [37] and can be cached. After retrieval, only a concise payload (a few short texts or embeddings) is fed to the model, so the prompt length remains small. Thus, prefill latency stays low. One potential drawback is complexity – a whole retrieval system must be maintained – and the risk of missing relevant context if retrieval fails. But when done well, this approach **avoids any per-user increase in model footprint or context length**, scaling personalization to millions of users by leveraging fast lookup of stored representations.

In summary, the field is converging on *personalization techniques that inject compact latent representations of user context* instead of raw lengthy prompts. Soft prompts and prefix-based methods offer simplicity and speed, while encoder-based compression improves capacity to handle rich histories [25] [24]. Advanced architectures like cross-attention memory further enhance expressiveness at some cost of complexity. Retrieval-enhanced strategies add modularity, letting systems personalize deeply without bloating the core model. The optimal solution often combines ideas: for instance, a production system might use a retrieval step to curate a user's context, an encoder to compress it into a soft prefix, and a frozen 4B model to generate the output. All these approaches share the goal of **minimizing per-user overhead at inference** – both in tokens and in model parameters – enabling personalized AI experiences that are scalable and fast. As research continues, we expect even more efficient hybrids (e.g. dynamic prompt generators, implicit memory in LLMs, etc.) to push the frontier of **low-latency, high-quality personalization** [11] [35].

Sources:

- Li & Liang (2021), Lester et al. (2021) – Prefix and prompt tuning for LMs [38] [4]
- Lester et al. (2021) – Efficacy of prompt tuning at scale [2]
- Hebert et al. (2024) – *PERSOMA: Personalized Soft Prompt Adapter* [7] [20]
- Liu et al. (2024) – *LLMs + Persona-Plug = Personalized LLMs* [24] [23]
- Salemi et al. (2023) – Personalization via LLM + memory (LaMP benchmark) [39] [24]
- Lee et al. (2023), Zhang et al. (2025) – Studies on injecting personas into LLMs [40] [41]
- Liu et al. (2024) – PersonaPlug user embedder approach [23]
- He et al. (2024) – PERSOMA architecture and results [7] [21]
- Anon (2025) – *Memory-Augmented LLMs Survey* (external memory via prefix vs cross-attn) [27]
- Shinwari et al. (2025) – Long-term memory interface for LLM (cross-attention integration) [42] [27]
- Zhaoxuan Tan et al. (2024) – *Personalized Pieces (PER-PCS)* adapter sharing framework [33] [35]
- Grattachi et al. (2024) – E2P: Embedding-to-Prefix method and deployment considerations [11] [17].

1 3 5 6 9 10 11 12 13 14 15 16 17 38 39 40 41 Embedding-to-Prefix: Parameter-Efficient Personalization for Pre-Trained Large Language Models
https://arxiv.org/html/2505.17051v1

2 4 Parameter-Efficient Language Models
https://www.emergentmind.com/topics/parameter-efficient-language-models

7 8 18 19 20 21 22 25 PERSOMA: Personalized Soft Prompt Adapter
https://www.emergentmind.com/papers/2408.00960

23 24 [2409.11901] LLMs + Persona-Plug = Personalized LLMs
https://arxiv.org/abs/2409.11901

26 [PDF] Flamingo: a Visual Language Model for Few-Shot Learning - NeurIPS
https://proceedings.neurips.cc/paper_files/paper/2022/file/960a172bc7fbf0177ccccbb411a7d800-Paper-Conference.pdf

27 28 29 30 36 37 42 Memory-Augmented LLMs: Enhanced Context Recall
https://www.emergentmind.com/topics/memory-augmented-large-language-models-llms

31 DeepMind Flamingo Explained — 32 Images are Enough To Learn!
https://pub.towardsai.net/deepmind-flamingo-explained-32-images-are-enough-to-learn-8badb73d1efc

32 Audio Flamingo 2: An Audio-Language Model with Long ... - Substack
https://substack.com/home/post/p-158725419?utm_campaign=post&utm_medium=web

33 34 35 aclanthology.org
https://aclanthology.org/2024.emnlp-main.371.pdf