Creating General User Models from Computer Use

Omar Shaikh
Stanford University
Stanford, USA

Shardul Sapkota
Stanford University
Stanford, USA

Shan Rizvi
Independent
New York City, USA

Eric Horvitz
Microsoft Research
Redmond, USA

Joon Sung Park
Stanford University
Stanford, USA

Diyi Yang
Stanford University
Stanford, USA
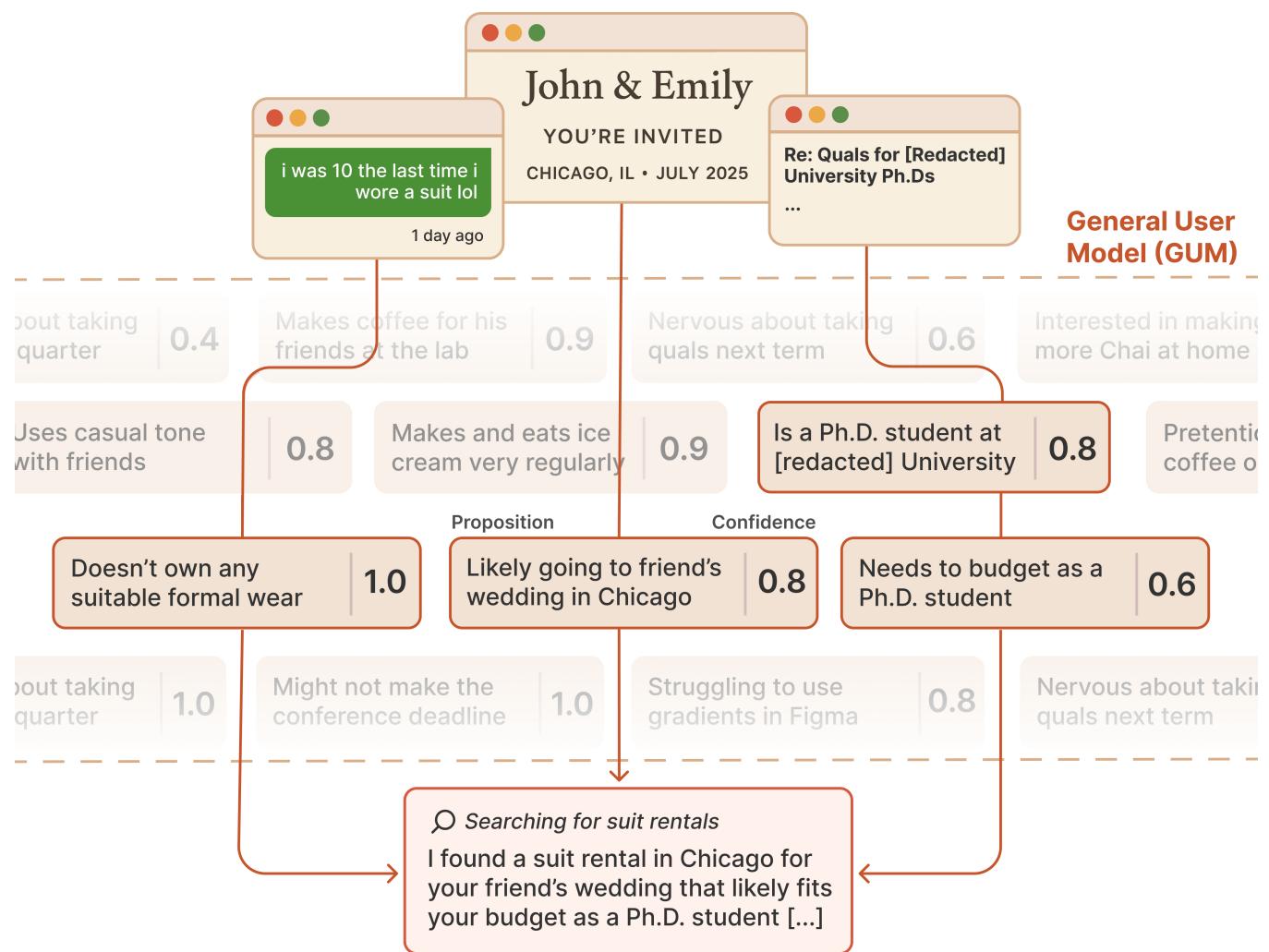
Michael S. Bernstein
Stanford University
Stanford, USA

Figure 1: General User Models take as input completely unstructured interaction data (top), transforming these observations into structured propositions about a user (center). Together, these propositions create a *general user model* (GUM). We instantiate our user model in an assistant (GUMBO) that proactively discovers and executes suggestions on a user's behalf (bottom).

Abstract

Human-computer interaction has long imagined technology that understands us—from our preferences and habits, to the timing and purpose of our everyday actions. Yet current user models remain fragmented, narrowly tailored to specific applications, and incapable of the flexible, cross-context reasoning required to fulfill these visions. This paper presents an architecture for a *general user model (GUM)* that learns about you by observing any interaction you have with your computer. The GUM takes as input any unstructured observation of a user (e.g., device screenshots) and constructs confidence-weighted natural language propositions that capture that user's behavior, knowledge, beliefs, and preferences. GUMs can infer that a user is `preparing for a wedding they're attending` from a message thread with a friend. Or recognize that a user is `struggling with a collaborator's feedback on a draft paper` by observing multiple stalled edits and a switch to reading related work. GUMs introduce an architecture that infers new propositions about a user from multimodal observations, retrieves related propositions for context, and continuously revises existing propositions. To illustrate the breadth of applications that GUMs enable, we demonstrate how they augment chat-based assistants with contextual understanding, manage OS notifications to surface important information only when needed, and enable interactive agents that adapt to user preferences across applications. We also instantiate a new class of proactive assistants (Gumbos) that discover and execute useful suggestions on a user's behalf based on the their GUM. In our evaluations, we find that GUMs make calibrated and accurate inferences about users, and that assistants built on GUMs proactively identify and perform actions of meaningful value that users wouldn't think to request explicitly. Altogether, GUMs introduce new methods that leverage large multimodal models to understand unstructured user context, enabling both long-standing visions of HCI and entirely new interactive systems that anticipate user needs.[1]

CCS Concepts

• **Computing methodologies → Natural language processing**; • **Human-centered computing → Interactive systems and tools**.

Keywords

User models, natural language processing

1 Introduction

In our most celebrated visions of human-computer interaction, interactive systems deploy a rich understanding of our goals, tasks, and context. Mark Weiser's central scenario in *The Computer For the 21st Century* opens with an alarm clock that knows when its owner is about to wake, and proactively offers coffee [74]. Likewise, Apple's Knowledge Navigator looks up useful supporting information while its user is puzzling over a question, then blocks an undesired phone call while the user is focused [5]. This notion of technology that *knows the user well enough to do the right thing at the right time* weaves through visions of both context-aware systems [18],

---

[1] A demo of Gumbo and our open source package for GUM are available at https://generalusermodels.github.io

which promise to adapt as the user's situation evolves, and interface agents [47], which take proactive action on the user's behalf.

Today, however, these visions remain largely out of reach. Despite progress in user modeling, recommender systems, and context-aware systems, computers remain remarkably unaware of who we are, what we are doing, and what would be helpful. At their core, current user models are simply too *narrow*: they understand our music preferences, our use of tools within a single app, or the TV show we might watch next. Even when user models integrate data across multiple applications, the integration remains surface-level; user models cannot reason or make inferences in new contexts. Our visions of technology, however, require user models that are *broad*, able to reason about everything from our general preferences to our immediate information needs; and capable of applying these insights across contexts, from work-related tasks to recreational activities. Applications today stumble because they have pinhole views of users: Weiser's vision of ubiquitous computing requires models that reason about family, friends, and work—not one application, and not just via a one-dimensional signal.

In this paper, we describe an architecture for **general user models**: computational models that materialize information and inferences about a user across domains and time scales. The general user model, or **GUM**, allows a user to construct a private, computational representation of their own behavior, knowledge, beliefs, and preferences by feeding unstructured observations (e.g., screenshares of their computer use) through an inference architecture leveraging large multimodal models (e.g., vision and language models, or VLMs). Our architecture contributes three main elements. A *Propose* module translates unstructured observations into confidence-weighted propositions about the user's preferences, context, and intent. A *Retrieve* module indexes and searches these propositions to return the most contextually relevant subset for a given query. Finally, using results from Retrieve, a *Revise* module reevaluates and refines propositions as new observations arrive. We audit all observations for privacy violations with a contextual integrity [50] *Audit* module that leverages the GUM itself to estimate and filter out information that the user would not expect to be recorded into the GUM. All data stays securely on the user's device, enabling local inference on capable hardware.

The operating system, applications, or the user themselves can query the GUM in real time to realize a breadth of applications similar to those envisioned in foundational HCI work. As part of the GUM, we introduce an interface that enables applications to query the GUM for underlying propositions. Any unstructured observation that the GUM sees can be marshalled to power interactive applications. Regardless of the interaction, users maintain direct and local control of the GUM's underlying propositions, allowing for edits, deletions, or additions.

At the simplest level, the GUM can insert information to establish common ground between applications and a user: for example, automatically adding relevant context when prompting a language model such as ChatGPT. With a GUM, any LLM can now directly reference the research paper you were reading just minutes earlier when you ask about its methodology, eliminating the need for you to explicitly quote or summarize the paper's content. Beyond prompting LLMs, any application can directly query the GUM to adapt their experiences, realizing long-standing HCI visions. A

GUM-enhanced operating system, for example, could prioritize only truly relevant notifications during a meeting—surfacing an imminent conference registration deadline while suppressing recipe emails. Email clients connected to a GUM could automatically sort messages based on observed user priorities, without requiring additional application-specific training.

GUMs also enable the creation of an entirely new class of proactive, interactive systems. We demonstrate this through an assistant, GUMBO, that learns a GUM via continuous private screenshot capture of the user's computer. Using the GUM, GUMBO constantly discovers *what* suggestions would be helpful to surface conditioned on the user's context. In addition, GUMBO uses the underlying GUM to determine *if and when* it might be useful to intervene with, and autonomosuly execute on, a suggestion. By marshalling a user's context, GUMBO can proactively discover a range of useful suggestions and filter important ones appropriately.

For the first author of this paper, GUMBO proactively found a location to rent a suit after observing a wedding invite from their friend (constrained by the author's budget, see Fig 1). GUMBO also found and proposed fixes for bugs in the system *itself* during development; and suggested potential revisions to this paper based on interactions with collaborators. For participants in our evaluation, GUMBO brainstormed ways to integrate a new theoretical framework into ongoing research, created a highly personalized moving plan for a cross-country relocation, and helped organize email archives from scattered correspondence—all proactively, based solely on its observations of users.

In our technical evaluations, we first focus on validating GUM accuracy. We train GUM on recent email interaction, feeding each email—metadata, attachments, links, and replies—sequentially into the GUM. $N = 18$ participants judged propositions generated by GUMs as overall accurate and well-calibrated: unconfident when incorrect, and confident when correct. Highly confident propositions (confidence = 10) were rated 100% accurate, while all propositions on average—including ones with low confidence—were fairly accurate (76.15%). From ablation studies, we show that all GUM components are critical for accuracy. We then deploy GUMBO with $N = 5$ participants for 5 days, with the system observing the participants' screens. This longitudinal evaluation replicated our results with the underlying GUM. Additionally, participants identified a meaningful number of useful and well-executed suggestions completed by GUMBO. Two of the five participants found particularly high value in the system and asked to continue running it on their computer after the study concluded. Our evaluations also highlight limitations and boundary conditions of GUM and GUMBO, including privacy considerations and overly candid propositions.

In sum, we contribute *General User Models (GUMs)*: computational representations of a user's behavior, knowledge, beliefs, and preferences, built from unstructured observations of the user. We demonstrate an implementation of a GUM, an interface allowing applications to query the GUM, an example assistant application called GUMBO, a technical evaluation via unstructured email interactions, a longitudinal evaluation via unstructured screen captures, and a reflection on norms and implications of this class of application.

2 Related Work

The first major challenge that GUMs must engage with is common ground—the mutual knowledge, beliefs, and assumptions shared between individuals—which enables efficient communication [16]. Through dialogue, gestures, facial expressions, eye gaze, and other multimodal cues [14], people continually update their shared mental models, iteratively constructing common ground [15]. Common ground acts as a shared context, allowing people to communicate without much unnecessary elaboration.

However, AI systems have little or zero shared context with users and make assumptions about a user's background, producing one-size-fits-all answers that are over-informative [70], overconfident [48], and unable to handle ambiguity [1, 30, 49]. Inability to ground extends beyond question-asking. For example, designing and prototyping with autonomous AI systems necessitates collaborative grounding [71]. Otherwise, they might take unfavorable and irreversible action [8, 12, 66].

To bridge the human-AI grounding gap, two general solutions have emerged: asking clarification questions or learning from a user's dialogue history. Clarification-based grounding strategies attempt to simulate the natural back-and-forth dialogue that humans use to establish shared understanding. This interaction can be implemented through prompting [13, 44], finetuning [4, 29, 34, 81], or a combination of both [64]. Learning from or keeping a memory of a user's chat interaction history is another solution [43, 46, 56, 63, 82]. In practice, current models are limited to context derived solely from their own prior interactions with the user. All other computer interaction is inaccessible to models.

We argue that engaging in grounding only when a user interacts with a model is a constraining interaction paradigm. Piecing together custom grounding interactions for specific domains—and fixing these interactions to dialogue—limits the ubiquity of potential common ground users can build with AI systems. In this work, we propose *general* user models that proactively build common ground with users through observation.

The second major issue that GUMs draw on is that prompting LLMs effectively is difficult [79]. Users often provide models with underspecified prompts [2, 64, 78], requiring repeated iteration to specify constraints for a specific context. A range of interactive systems have enabled users to better specify constraints to LLMs given a specific context: systems like ChainForge [6] and EvalGen [65] offer users a means to interactively validate and iterate on prompts for specific tasks. Beyond developing prompts, systems like Graphologue [41] and Sensescape [68] allow users to explore the range of possible outputs given a prompt. Across these interactions, the burden of providing explicit input to build a shared context is largely on the user.

In contrast, a longstanding goal in HCI is to build systems that proactively assist users without disruption. Early work in mixed-initiative interfaces balanced autonomy with user control, leveraging context to identify the best moments to interact [35]. Classic AI models have already been employed to understand a user's multimodal context. Memory Landmarks and the LifeBrowser system [37], for example, construct an understanding of which events users find important [58] by analyzing their calendar and mailbox. A handful of systems have started using LLMs to selectively process

context. PICAN [33], for example, focuses on building context in VR-onboarding settings by focusing on dialogue and actions the user takes in the metaverse. Systems like GPTCoach [42] rely on external health data and qualitative context, augmenting LLMs with selective context to help with behavior change.

Still, our interactions with LLMs are far from visions like Weiser's Sal, where computers unify machine intelligence with context from our daily lives [36]. Interactive LLM-based systems focus on *constraining* the types of inputs, outputs, and tasks these models operate over. In this paper, we embrace what LLMs are good at—unconstrained input—and ground assumptions about the user and their context based on observation. We design what is effectively a multimodal grounding engine.

Context aware computing has long envisioned such toolkits [18, 20, 61]. Yet these frameworks are typically pinned down to some underlying fixed *structure* of the input data or the underlying sensors [23]. This inevitably limits how we define context and the types of applications we can support from the outset. GUMs offer an alternative approach—they introduce a flexible structure with very few constraints to build context.

3 General User Models

In this section, we introduce an architecture for learning general user models. GUMs enable learning both stable inferences about the end user and moment-to-moment context, using everyday action as input data. From low-level input (e.g. screenshots), GUMs construct a rich understanding of who we are and what we are doing. Applications can then query the GUM to power a wide range of interactions, from operating systems that power contextual notifications to assistants that proactively discover and execute useful suggestions based on context.

At its core, GUM is a collection of natural language, confidence-weighted propositions about a user, learned entirely from observation. The following propositions, for example, were constructed by the author's GUM while they wrote this section. Each proposition is also paired with a confidence score (from 0 to 1).

> **Proposition**: `[Anon]` is currently writing in the General User Model's section.
> **Confidence**: 0.8
>
> **Proposition**: `[Anon]` is viewing and resolving comments from their advisor, `[Anon]`.
> **Confidence**: 0.7
>
> **Proposition**: `[Anon]` is struggling with the technical evaluation of GUMs.
> **Confidence**: 0.5

These propositions were generated entirely using a series of screenshots from the author's screen. To construct propositions, GUM takes any type of input that an underlying large model can accept (in the case of vision-langauge models, GUM accepts image and text input). Using this unstructured input, GUM composes and refines proposition-confidence pairs. This abstraction—a series of confidence-weighed statements that conjecture some *common ground* between the user and their computer—enables a flexible

representation of a user's context. To be clear: GUMs do not enforce any structure on the types or kinds of propositions that can be generated. Propositions are constrained only by natural language descriptions, and are revised as the GUM continues receiving more input. GUMs aim to continually learn this representation of a user's context from any interaction.

Here, we detail abstractions (confidence-weighted propositions) that power the GUM (§3.1) and document an interface for using GUMs in other applications (§3.2).

3.1 Propositions and Confidences: Abstractions for Representing General User Models

GUM draws inspiration from literature on common ground in human-human interaction [14, 15]. A subset of common ground between participants can be defined by shared assumptions. The human-AI grounding challenge lies in identifying *valid* assumptions an AI model can make about a user. When constructing a user model, we must include shared assumptions that accurately represent user actions. For example, when a user repeatedly searches for "best hiking trails in Colorado" and visits outdoor equipment websites, the model might form the proposition "User enjoys outdoor activities, and is interested in hiking in Colorado" with high confidence. Similarly, if a user has numerous calendar entries for guitar lessons and frequently browses music theory websites, the model might infer "User is learning to play guitar and is interested in music theory" as a shared assumption.

We build abstractions for GUM based on models of grounding in human-human communication [10]. Bayesian approaches represent shared knowledge as probabilistic distributions, not deterministic sets of propositions. In these models, interlocutors maintain probability distributions over possible states, refining a shared understanding through communication [10]. This probabilistic view accommodates user uncertainty and grounding over time [38, 45]. To instantiate a probabilistic model of common ground, we leverage the in-context learning capabilities of large langauge models, which can be loosely interpreted as Bayesian updates [76, 80]. We operationalize these approaches through three abstractions: observations, propositions, and confidences.

Observations are the raw inputs that the GUM processes. Observations can be anything, so long as it can be tokenized and fed into a large (vision/language/multimodal) model. Screenshot observations provide visual data of what the user is viewing, file system observations reveal document interactions, and notification observations offer communication pattern insights. Observations are the source from which GUM builds user context understanding. Each observation includes metadata: the source, timestamp, and a unique identifier. Unlike propositions, observations are factual records, not inferences. The relationship between observations and propositions is many-to-many. Multiple observations might support a single proposition, and a single observation might contribute to multiple propositions.

Propositions are the assumptions, in text, that GUM constructs through observation. Viewing a friend's wedding invite yields surface inferences like `User is invited to friend's wedding`. However, GUM may generate more complex inferences requiring speculation, e.g., `User needs to buy or rent a suit for`

the `wedding`. These propositions might not be correct, making uncertainty inclusion critical. We, therefore, include confidences alongside each proposition.

Confidences reflect how certain the model is about proposition truth, predicated on the quality and quantity of the available evidence to support the proposition. Confidences enable decision making over uncertainty, representing belief degrees associated with each proposition. This allows prioritizing actions based on reliable information while acknowledging tentative assumptions. `User is invited to friend's wedding` requires less speculation: the invite appears on-screen, earning high confidence (1.0). Inferences about `needing to buy or rent a suit` are never explicitly expressed, receiving lower confidence (0.4). In §6, we evaluate how well the GUM's generated confidence scores align with participants' actual accuracy.

Additionally, propositions include **metadata** for downstream applications. Inspired by cognitive agent architectures [3, 57], we introduce generated **decay** factors (0-1), indicating proposition staleness rates. `Author is a Ph.D. student` decays slowly (factor = 1.0). Conversely, `Author is debugging a memory leak` is transient (factor = 0.6). Each proposition includes **grounding**: observations supporting the inference, plus generated **reasoning** explaining the proposition's construction:

> Screenshots show the author using XCode's memory profiler, examining increasing memory consumption graphs, inspecting code functions with memory allocation annotations, and adding debugging statements around object creation - all indicating active memory leak troubleshooting.

3.2 Interfacing with the GUM

Here, we outline core functions to interface with the GUM, highlighting how applications and users can interact with the underlying abstractions. GUMs are implemented in Python can be pip-installed as a package. Later in §5, we revisit how GUM implements each of these functionalities.

Instantiating a GUM. Instantiating a GUM requires access to an instruction-following (V)LM (e.g., GPT, Claude, Llama). We use Llama 3.3 70B running on secure servers to maintain privacy, though any model supporting the OpenAI ChatCompletions API is compatible. All propositions and associated metadata are located under the private `GUM._propositions` variable, accessible only to the user. Applications cannot directly manipulate the private `propositions` variable attached to the GUM, unless explicitly granted permission by the user.

Subscribing to observations. The GUM can `subscribe` to various information streams with user permission. Like traditional context-aware systems that require operating system hooks [24], GUM also exposes a handful of default hooks, which we call Observers. These Observers listen for new observations related to screen content, notifications, and manual input. For example, when a user saves a file or moves their mouse, the Filesystem and Screen Observers pass new observations to the GUM, which converts these observations into propositions.

Help me with this section

🔍 *Retrieving from General User Model*

Writing a research paper on General User Models	1.0
Writing the "Applications of General User Models" section	1.0
Regularly references Weiser and Maes et al. for inspiration	0.8
Thinks current applications of LLMs are too constrained	0.8

Since you're revising the Applications section and have been thinking about Weiser's Calm Computing, it might be helpful to emphasize how your system extends calm computing principles. Here's a paragraph I drafted...

Figure 2: General User Models can ground user queries in a broader context. In this example, the user is writing an Applications section for this paper. Given a severely underspecified query, the General User Model marshalls relevant context—the draft the user has been working on, the section the user is referring to, the applications the user has already considered—and generates a highly contextualized response.

Querying the GUM. Users and applications alike can search for propositions (and underlying observations) using natural language queries. The query function supports parameters for search terms, response diversity, and time filtering. For example, searching for "HCI conference" will return relevant propositions about the user's HCI-related activities. The query function also supports options to control a handful of parameters: the balance between the relevance and diversity of responses (0 = maximize relevance, 1 = maximize diversity), the timestamp cutoff from which the query should start its search (looking back), and whether or not a decay should be applied based on the proposition's timestamp.

4 Applications of General User Models

To illustrate the breadth of applications for a GUM, we outline a range of downstream use cases. These span from simple applications, such as augmenting powering chat-based LLM clients with context from GUM, to introducing a new class of proactive systems that discover helpful suggestions for users. We also reflect on how GUMs empower classic HCI visions, including ubiquitous computing, interface agents, and context-aware computing.

4.1 Prompting Models

Before we get distracted by visions, let's return to something slightly more humble: prompting. A fundamental problem with prompting is a lack of shared context—called *grounding* [64]—with a model. Unless explicitly instructed, LLMs have no idea what project you're working on, who you're working with, or why the prompted task was important to you. This grounding gap results in models that often incorrectly *assume* aspects of our context.

Prompting LLMs is inherently underspecified: we're forced to turn our rich context into a decontextualized natural language query. Here, we show how GUM can help close the *grounding gap* [62] in how we interact with chat-based LLMs.

Consider the following prompt alongside the author's own GUM at this point in the paper (see Fig. 2).

> USER: help me with this section.

Today's frontier LLM models are lost, returning a clarification question ("Sure, I can help! Could you share the section you're referring to?"), even though there's a rich amount of context to be constructed from observation. The author's GUM has access to interactions between the authors, papers the author has recently been reading, the state of the current paper, prior HCI work the author likes, etc. In contrast, with a standard LLM users must (at worst) painfully rebuild this context turn by turn, correcting the model when it incorrectly makes assumptions. At best, LLMs rely on context from *past* chat interaction (using a sliver of the user's true context) or ask clarification questions.

Here, the GUM bridges this grounding gap by querying for additional propositions and their underlying observations. In our example, the GUM returns papers the author was looking at earlier, feedback from the coauthors, and the section the author was revising.

> **Proposition**: `[Anon] is currently focusing`
> `on the Applications section.`
> **Confidence**: 0.8
>
> **Proposition**: `[Anon] recently read Weiser's`
> `Calm Computing.`
> **Confidence**: 0.6
>
> **Proposition**: `[Anon]'s advisor mentioning looking`
> `into Interface Agents.`
> **Confidence**: 0.5

Conditioned on the GUM, LLMs generate responses that are far more personalized to the specific user.

> Since you're revising the Applications section and have been thinking about Weiser's Calm Computing and Interface Agents, it might be helpful to emphasize how your system extends calm computing principles through contextual awareness. You could also contrast GUMBO's general assistance with the more fixed approach seen in traditional interface agents. Here's a paragraph I drafted...

4.2 Any Application Can Query A GUM

Beyond prompting, any application can connect to a GUM, using context from all *other* applications and the user. Across a range of HCI systems, having access to context and being able to process it is a key single enabling layer of the application. To outline this, we revisit Weiser [74]'s calm technology, Maes and Kozierok [47]'s interface agents, and, in a slightly "meta" sense, Dey et al. [20]'s context protocols.

Calm Technology (Weiser and Brown [75]). Weiser's ubiquitous computing vision launched a raft of work realizing systems that slide to and from the user's periphery. People have typically built such systems based on custom sensors [27, 39] such as cameras and microphones, where each application has to build a custom ML pipeline to make decisions [28, 38]. The GUM enables all of these applications with no such instrumentation or training set, in a way that is sensitive to the user's task. Weiser's calm technology engages both the center and periphery of a user's attention, moving seamlessly between the two. A key design consideration for calm technologies [75] is recognizing that what resides in the periphery can shift from moment to moment.

Applications of GUMs realize this vision. Operating systems that use GUMs can surface important information only when it's truly needed, avoiding information overload. For example, the first author of this paper was swamped with writing this paper, and the CHI early bird registration deadline came and went. By default, their operating system shows notifications for every email they gets in their primary inbox—causing them to generally ignore the notification, as they also received emails in the same period about "Ice Cream Recipe Found!", "firebase-noreply-billing", and "Job Opportunity at Startup". When prompting Llama 3.3 with "Filter out notifications that are important to me.", it suggests passing all of these notifications through to the author's attention, leading to our poor author's attention overload. However, the same prompt, augmented with GUM's propositions and confidence, provides context that the author (1) "is quite happy as a Ph.D. student", (2) "has a major impending deadline", (3) "regularly ignores billing notifications", and (4) "makes ice-cream on weekends as a hobby." As a result, the LLM now allows only the CHI registration notification through.

Interface Agents (Maes and Kozierok [47]). Interface agents introduced personalized and proactive assistance as a vision for human-computer interaction. Instead of placing the burden of initiative on the user, interface agents are applications that aim to anticipate user needs and automate tasks. This requires both domain knowledge and personalized user models, but an implementation has remained challenging without a continuously learned model of individual preferences. Each new agent requires specialized datasets and hand-crafted learning objectives to capture evolving user preferences [47].

GUMs enable interface agents more broadly. They supply both the user model and reflect the user's preferences across time. Consider Maes' Firefly agent [67]: a movie recommendation agent that proactively surfaces interesting new movies based on the user's preferences. For the author, now would be a horrible time—they are scrambling to finish their paper. But when the author prompted a standard LLM (Llama 3.3 70B) and asked if now was a good time for

a movie, it agreed *and* proceeded to recommend a list of movies that were completely unrelated to the author's interests (psychological thrillers). In contrast, with the same prompt, the GUM-augmented LLM recognizes that the author (1) has "many comments unresolved on his draft" and (2) has only "a few hours until the paper deadline". As a demonstration, when we ablate everything deadline-related from the GUM, the GUM-augmented LLM correctly agrees and recommendes a relevant list of movies. With GUMs, we can instantiate a range of proactive personalized interface-agents, with minimal domain-specific engineering and a handful of simple prompts (e.g. `Should I watch a movie?`).

Context-Aware Computing. Unifying context across applications is a long-standing vision of context-aware computing, but creating effective context protocols [20] has remained evasive. If this were achieved, a common API could then be queried by any application to adapt based on relevant user context. Applications built using context toolkits—from context-aware reminder systems [19] to assistants for conference attendees [21]—relied on infrastructure for specialized sensors. Supporting a new *specialized* sensor is a challenge: for each new sensor, an entirely new data processing pipeline must be manually engineered.

GUMs promise to solve exactly this! Any input data type—a screenshot, sensor data, videos, etc.—can, in principle, be embedded in the GUM. As an example, the first author exported their self-reported physical exercise logs from a weight training app on their phone and serialized the day-level logs into text files. In addition, they exported step counts from their Apple Health app. With no data preprocessing (beyond initial .txt translation), the first author fed these observations directly into their GUM. The result:

> **Proposition:** `[Anon] is prioritizing their paper deadline over routine physical exercise.`
> **Confidence:** 0.6

4.3 GUMBO: an Assistant Built On GUMs

To more concretely ground the examples above, we now describe how GUMs enable a class of applications that can proactively discover suggestions based on user context. We introduce GUMBO (Fig. 3), an assistant that ingests screenshots from a user's screen. Using screenshots, GUMBO builds an internal GUM. With the GUM, GUMBO discovers helpful suggestions, determines if a suggestion is worth showing to a user and executing, and then executes the (sandboxed) suggestion to the best of its ability—sharing preliminary results with the user.

GUMBO demonstrates how to construct a system that utilizes GUM in many different capacities simultaneously: (1) generating rapid, relevant suggestions that the system can do on behalf of the user or help them complete; (2) estimating how much utility each suggestion will provide to the user if completed, enabling Horvitz's mixed initiative interaction framework [35]; (3) revising the model with explicit user feedback.

4.3.1 Discovering Suggestions. Before suggesting anything to a user, we must generate a candidate set to evaluate. GUMBO retrieves the latest propositions from GUM and uses them to generate candidate suggestions. It then employs GUM to rank and filter these suggestions, presenting only a curated subset to the user.

When a new proposition is constructed, e.g., `User is likely going to friend's wedding in Chicago`, GUMBO retrieves related propositions G using the query function provided by GUM (`User doesn't own any suitable formal wear` from recently browsing clothing sites, `User needs to budget based on recent bank account checks`, etc.). These propositions are pulled based on semantic relevance to "wedding travel" and the recency of related observations (e.g. pulling up a bank statement or starting a search for buying formal wear online).

Using the set of related propositions G, along with all the underlying observations attached to each proposition, we generate a set of candidate suggestions $\tau_i \sim P_{LM}(* \mid G)$ (e.g. `Search for cheap suit rentals in Chicago`). To generate candidates, we prompt the underlying LLM, Llama 3.3 70B, with all the propositions and underlying metadata. An abridged prompt is below:

```
{raw observations omitted}
```

> **Proposition**: `[Anon] is likely going to friend's wedding in Chicago.`
> **Confidence**: 0.8

> **Proposition**: `[Anon] doesn't own any suitable formal wear.`
> **Confidence**: 0.6

> **Proposition**: `[Anon] needs to budget as a Ph.D. student.`
> **Confidence**: 0.5

```
What concrete suggestions do you have for
the user based on the provided context?
```

For each suggestion, our full prompt also elicits additional confidence in the generated suggestion being something the user would find any value in. This can be interpreted as an aggregate suggestion confidence ($P(\tau_i \mid G)$), which will come in handy when we decide if it's worth showing a user τ_i. For each new proposition, we use the prompt above to generate five suggestions together.

4.3.2 Determining if suggestions are worth showing. GUMBO can generate and suggest a slew of suggestions that are relevant to the user's context. In many cases, suggestions are repeats—we simply filter using lexical overlap heuristics. The real challenge arises when we have many different, potentially useful, suggestions.

We apply mixed-initiative interaction [35] principles to balance helpfulness against intrusiveness. This approach requires estimating both the probability of a suggestion being useful and its utility to the user. By calculating the expected utility of interruption versus non-interruption, we can make informed decisions about when to surface suggestions. We weigh the probability the user would find some value in the suggestion $P(\tau_i \mid G)$ against the benefits of suggestion completion B and the costs of false positives C_{FP} and false negatives C_{FN}.

Mixed initiative interaction assumes some model that produces costs and benefits for a specific suggestion—a major cold start hurdle for the practical implementation of mixed-initiative systems to date. In our case, we turn again to the GUM, which can estimate
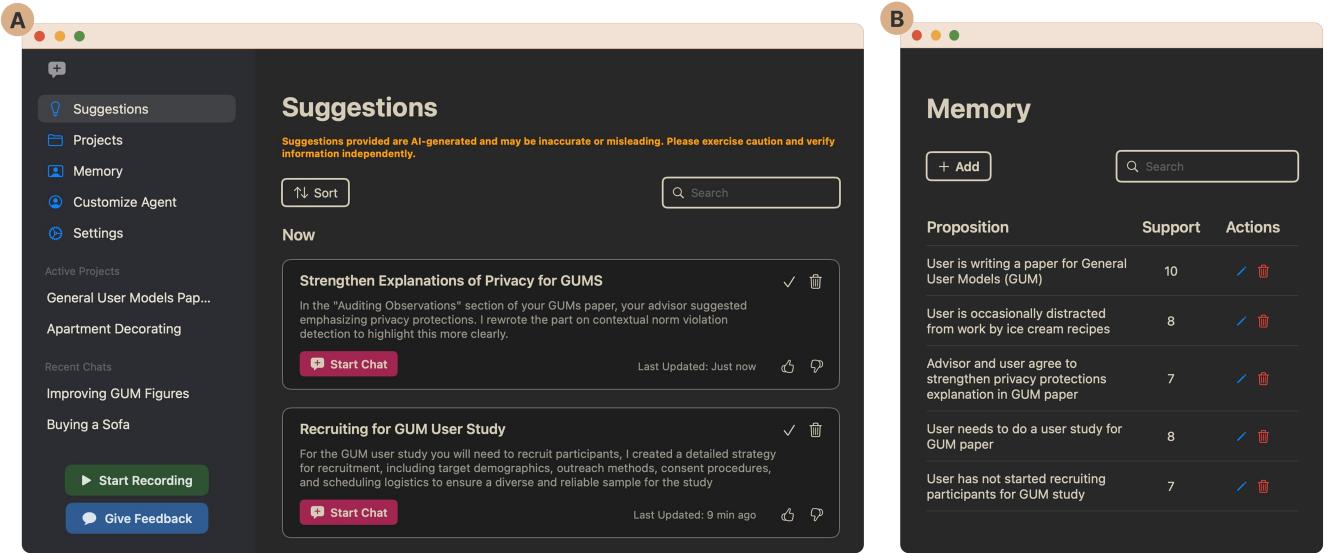
Figure 3: GUMBO is an assistant designed to proactively surface and execute helpful suggestions for users. We built GUM into a custom desktop app. (A) The Suggestions page displays suggestions to the user, each of which the system has attempted to complete as much as possible on its own. In this figure, GUMBO suggested that the author recruit participants for the GUM user study and put together a detailed strategy plan in the background. Users can hit "Start Chat" to talk to GUMBO in more detail. (B) The Memory page allows users to view the raw propositions in their GUM, and edit, delete, or add propositions.

these quantities with no further training. Conditioned on the propositions from the GUM, we elicit costs and benefits using a prompt (Appendix B.2.1)—similar to how we generated suggestions. With the costs, benefits, and probabilities, we can compute the expected utility of (not) interrupting ($E[U]$) the user with the suggestion:

$$E[U_{\text{interrupt}}] = P(\tau_i \mid G) \cdot B \ + \ (1 - P(\tau_i \mid G)) \cdot (-C_{FP}). \quad (1)$$

$$E[U_{\neg\text{interrupt}}] = P(\tau_i \mid G) \cdot (-C_{FN}) \ + \ (1 - P(\tau_i \mid G)) \cdot 0, \quad (2)$$

$$= P(\tau_i \mid G) \cdot (-C_{FN}). \quad (3)$$

To determine a decision threshold, we can simply test if the expected utility of interrupting is greater than not interrupting.

$$E[U_{\text{interrupt}}] > E[U_{\neg\text{interrupt}}] \quad (4)$$

Occasionally, we found that suggestions would still pour through the decision boundary. The issue arises because Horvitz [35]'s mixed-initiative framework treats actions as independent—here though, the utility of adding an additional suggestion depends on how many other suggestions are being presented to the user in the same time period. To address this, we implement an additional token-bucketing algorithim [17], rate-limiting suggested suggestions to a maximum of 1 per minute. Altogether, we can re-use GUM to determine when to interrupt in the first place, operationalizing principles from mixed initiative interaction.

4.3.3 Ideas are cheap. Execution is everything. Instead of solely suggesting that the user `Find a suitable place to rent a suit in Chicago`, GUMBO should go a step further and search for rental locations on its own and report its findings. In other words, if GUMBO is capable of completing the suggestion—and doing so does not cause irreversible side effects (such as actually ordering a suit)—it should execute the suggestion itself and present the results to the user.

When necessary, suggestions generated by GUMBO can be delegated to a set of tools or agents. We use an additional zero-shot prompt to determine if a suggestion requires a tool (see Appendix B.2.2). We implement a handful of external tools which can be toggled by the user (and are by default disabled) to preserve privacy. GUMBO delegates calls to Gemini 2.0 Flash *only* if search with Google or code execution is required. In addition, GUMBO implements file system search and retrieval through the local MacOS spotlight search, and computer use via the Operator API.[2]

4.3.4 Feedback. Finally, GUMBO allows users to leave thumbs up / down or natural language feedback on any suggestion. GUMBO simply converts the feedback into a text representation (`User disliked the following suggestion: [suggestion]`) and feeds it back into GUM. We treat feedback the same as any other unstructured observation, placing the onus on the GUM to convert feedback into appropriately weighted propositions.

5 Constructing a General User Model

To continuously learn a user model, GUMs *construct*, *retrieve* and *revise* propositions about a user by ingesting completely unstructured *observations*. In this section, we describe the architecture behind these components, and explain how they produce the GUM API.

---

[2]We leave Computer Use disabled in GUMBO's evaluation, both to reduce external dependencies and since we found computer use too slow / buggy. We expect advancements in computer use to greatly benefit from implementing GUMs
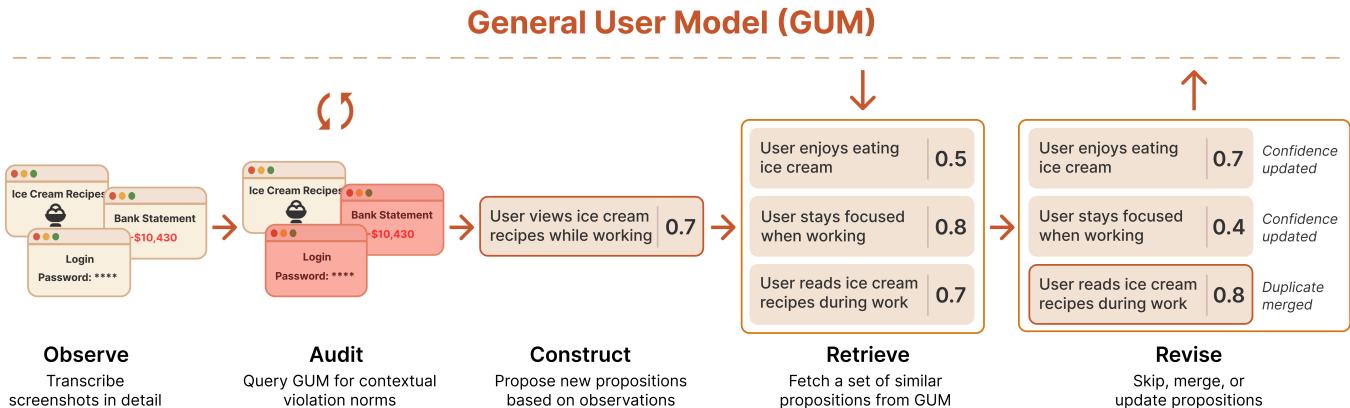
General User Model (GUM)

Figure 4: The GUM pipeline observes user behavior via unstructured inputs (e.g., screenshots), audits updates for privacy violations using contextual integrity, constructs propositions with associated confidence scores, retrieves contextually similar propositions, and revises the model. Each step iteratively refines GUM's understanding of the user.

Our general engineering principle here is to rely primarily on open-source models. While closed-source models are more performant [9], we expect GUMs to be *owned* by individual users and eventually distilled to be run on their local devices. By focusing on open-source models, we demonstrate and advocate for the core GUMs to operate without sending private user data to third party AI platforms. As gaps between closed and open sourced models close and as models become cheaper for inference, GUMs will become more performant and feasible on commodity hardware.

Powering the GUM in this paper are two models. To observe screen interactions, we use Qwen 2.5 VL [7]—a vision-language model with a vision encoder. For proposing/revising propositions, we use Llama 3.3 70B [31], a language-only decoder model.

5.1 Observing Interaction

The basic substrate of the GUM is user behavior. Where prior user models were often built off narrow, manually-logged observations (e.g., Fischer [25], Jameson [40], Horvitz et al. [38]), the inference abilities of modern VLMs open up opportunities for unstructured, unobtrusive inputs such as screenshots, files, or streams of open-ended text as the GUM's raw observations. The GUM Python package (introduced in §3.2) provides a handful of hooks that return raw observations. These hooks—Observers—are generic by design. For example, an audio Observer might connect to the device's microphone, transcribe the user's speech, and return that speech as a text string to the GUM—nothing else.

We implement a handful of Observer instantiations in Python for MacOS. The **Screen** Observer listens to I/O activity (keyboard, mouse clicks), captures a screenshot of the screen on I/O input, and transcribes the contents of the changes into a text update for the GUM. It relies on a VLM (Qwen 2.5 VL 72B) to convert screenshots into a transcript. Beyond the content, Screen also generates a description of actions the user takes across up to 10 unique frames (example outputs in Appendix E). A **Notification** Observer watches for updates on the operating system's underlying notification database and returns updates that contain the notification text and

contents. Our provided Observers—Screen and Notification—are just examples; one could implement new Observers for any modality, e.g. audio input, health sensor data, videos.

5.2 Auditing Observations

Even though GUM uses open-source vision-language models deployable locally, users may accidentally expose private information they never intended for processing. To prevent this, we implement an *Audit* module that filters listener updates before they reach the final collection of propositions.

GUMs learn a user's privacy expectations by observing data-sharing norms, using itself to infer user-specific models of contextual integrity. Consider two updates: (1) a user writing a work email about a conference versus (2) entering bank credentials on a financial website. The Audit module would allow the first as typical professional communication while blocking the second as sensitive financial data. In developing the Audit module, we instantiate Nissenbaum [50]'s contextual integrity theory, which preserves privacy when information flows match social context norms, focusing on appropriate sharing rather than binary data control.

Imagine a computer science professor's GUM. The GUM has already observed the user discussing course materials with colleagues via email and sharing research papers. However, it has never observed the user sharing login credentials or financial information with anyone. When the user enters their banking password on a financial website, the Audit module uses GUM to retrieve past propositions related to the current observation. These include:

```
[Anon] is careful about sharing access
credentials

[Anon] has discussed privacy concerns with
students in the past

[Anon] only discusses financial matters
with his spouse
```

Using these retrieved propositions, the Audit module then generates answers to Nissenbaum [50]'s contextual integrity questions:

> 1. Is the user disclosing any new information?
> **Yes**
>
> 2. What type of data is the user disclosing?
> **Banking credentials and financial account information.**
>
> 3. Who is the primary subject of the data?
> [User's name]
>
> 4. Who is the recipient?
> An AI model that ... [description of GUM]
>
> 5. Should this data be transmitted to the model?
> **No**

The Audit module determines this observation contains sensitive financial information that the user would not want shared, and blocks the update from progressing to the remaining pipeline.

5.3 Constructing Propositions

We now turn to making inferences about the user from observation. The Propose module converts observations into a set of propositions. During this process, Propose generates a reasoning trace that ties observations to a proposition and produces a confidence score associated with each proposition.

Reasoning About and Generating Propositions. Before generating a proposition, we want to ensure that the model reasons over *how* the observations are connected to the proposition. This rationale serves two purposes: it gives users some explanation for why an inference was made and it generally improves model performance [55, 72]. In this initial reasoning step, we first instruct the model to generate an explanation related to the observation's relationship to the user.

> observation: [screenshots of the user switching between Overleaf and YouTube]

With this observation, the *Propose* step generates the bolded **rationale**, explaining the user's behavior.

> proposition_reasoning: **"The user appears distracted, switching focus between an ice cream recipe video and typing intermittently in an Overleaf window."**

Once the rationale is generated, both the observation and rationale are jointly used to generate a final proposition, e.g. proposition $\sim P_{LM}(\cdot \mid \text{reasoning, obs})$

> proposition: **"User periodically views ice cream recipes while writing."**

This is true—the first author is an ice cream afficionado, and their preferred writing distraction is browsing ice cream recipes.

Producing Confidence Scores Over Propositions. Once the proposition and rationale are generated, we use GUM's underlying LLM to produce a confidence score in the proposition. Confidence scores

support future revisions of uncertain propositions and help applications make decisions under uncertainty. While we could, in theory, use the LLM's own log probabilities by looking directly at the logit scores on the completion , these predictions are often overly confident. Like Tian et al. [69], we observe that simply prompting the model to generate confidence scores yields more calibrated outputs instead of looking at underlying logits.[3] We prompt the model to generate a confidence score on a 1-10 scale, and then normalize between 0-1. Our full prompt is in Appendix B.1.1.

Proposition-specific decay. Some propositions remain more relevant over time compared to others. Unfortunately for the first author, a proposition like "User is eating ice cream." decays faster than "User is Ph.D. student." (Such is the nature of life. But, a proposition such as "User enjoys eating ice cream more than their Ph.D."[4]—that's timeless.) To account for this, we take inspiration from cognitive architectures [3, 57] and introduce a decay factor. Instead of applying a global decay to all propositions universally, we use the underlying LLM to generate a proposition-specific decay. We include all information about the proposition in-context (reasoning and confidence) and generate a decay score from 1-10, e.g. decay $\sim P_{LM}(\cdot \mid \text{confidence, proposition, reasoning, obs})$.

5.4 Retrieving Context-Sensitive Propositions

The process so far produces propositions and confidence scores (from Propose, §5.3), but we need to compare these with what GUM already knows. To enable comparison and retrieval over GUMs, we implement a *Retrieve* module. Speed is critical: each observation generates multiple propositions that require hundreds of queries. We use a two-stage **retrieve** and **rerank** approach [53]. First, we **retrieve** documents using BM25, which considers term frequency, inverse document frequency, and document length.[5] Then, we use an LLM-based **re-ranker** to improve precision. Our setup optimizes for speed but can accommodate other retrieval/reranking methods.

Recency-Adjusted Relevance. BM25 computes relevance scores, but recent propositions should be prioritized. Each proposition d_i includes a decay score α_i (§5.3). We define the decay factor as $\gamma_i = \exp(-\alpha_i \cdot k \cdot \text{age}(d_i))$, where $k = 2$ and $\text{age}(d_i)$ is a continuous value measured in days. The adjusted relevance score is:

$$\tilde{r}_i = r_i \cdot \gamma_i.$$

Incorporating Diversity via Document Similarity. Retrieved propositions should be diverse. We use Maximum Marginal Relevance (MMR) to balance relevance and diversity [11]. Using TF-IDF vectors for document representation, we compute cosine similarity between documents. Given a set S of selected propositions, the diversity term for a candidate document d_i is:

$$\delta_i = \begin{cases} \max_{d_j \in S} \text{sim}(d_i, d_j) & \text{if } S \neq \emptyset \end{cases}$$

The MMR score combines relevance and diversity:

$$\text{MMR}(d_i, S) = \lambda \tilde{r}_i - (1 - \lambda)\, \delta_i,$$

---

[3]Our setting is slightly different: we're looking at confidence in a user's beliefs, instead of confidence in the model's own response—but Tian et al. [69] generalizes.
[4]Yes, a real proposition generated by our system.
[5]While we use lexical similarity, one can easily replace our approach with neural embeddings

where $\lambda = 0.5$ balances the trade-off. We iteratively select documents maximizing MMR until reaching N documents.

Reranking and Filtering. We pass retrieved propositions through an LLM re-ranker that classifies them as *identical*, *similar*, or *unrelated* (the reranker prompt is in the Appendix B.1.2). For example, when querying with `User is periodically distracted by ice cream recipes`, a proposition like `User is a CS Ph.D. student` would be labeled as unrelated. On the other hand, `User might have a sweet tooth` would be related, while `User appears distracted by ice cream recipes` would be identical. Successive identical propositions might signal that a revision is necessary, which we address next in §5.5.

5.5 Revising Propositions

As the breadth and quantity of propositions in the GUM grows, revision becomes critical. The retrieve module simply surfaces similar propositions. However, newly generated propositions in GUM—flagged as *similar* by our retrieval module—can contradict or reinforce prior propositions. The final GUM should reflect these differences. We introduce a *Revision* module that takes newly generated propositions and appropriately revises prior ones. Over time, the revise step refines the GUM, yielding a collection of propositions that reflect a richer understanding of the user.

To implement revision, we construct a prompt that takes as input retrieved propositions and the newly generated propositions, along with the underlying observations for each proposition. The revision process can operate over multiple input/output propositions, revising multiple propositions at once. Using the newly generated propositions, the revision module rewrites the proposition, regenerates confidence, and then revises associated metadata. We encode both the past proposition(s) and the new proposition(s) in a prompt. We additionally include all metadata associated with both the old and new propositions—the underlying grounding, reasoning traces, and decay scores.

```
## Past Proposition
Proposition: User periodically views ice
cream recipes while writing.
Confidence: 6
Decay: 4
[additional metadata (grounding, reasoning)]

## New Proposition
Proposition: User messaged ice cream recipes
to colleague during a meeting.
Confidence: 6
Decay: 2
[additional metadata (grounding, reasoning)]
```

Conditioned on both the past and current proposition, we also generate a revised proposition: revision $\sim P_{LM}(\cdot \mid \text{prop}_{\text{new}}, \text{prop}_{\text{old}})$. Generated portions are **bolded**.

```
## Revised Proposition
[regenerated reasoning]
Proposition: User is regularly distracted
by ice cream recipes
```

```
Confidence: 10
Decay: 1
```

In the example above, our revised confidence increased. However, revision can also yield propositions with reduced confidence, especially when contradictory propositions are compared. We never completely evict propositions that are revised to zero confidence: they remain in the GUM for transparency. However, queries to the GUM will—by default—not surface confidence = 0 propositions.

6 Evaluating General User Models: Accuracy and Calibration

GUMs aim to accurately capture a user's context from unstructured observation. To first evaluate whether GUMs can effectively synthesize context into accurate inferences about users, we conduct an evaluation in a controlled setting using email data. We assess GUM's *accuracy* compared to ablations and evaluate GUM *calibration*—whether confidence levels appropriately reflect uncertainty.

We chose email as our evaluation domain because inboxes contain diverse and relevant information across modalities, while also being full of irrelevant content. Email overload makes synthesizing important information difficult: GUMs must maintain accuracy despite this challenge. Though email represents only a subset of user interactions, its controlled, semantically rich nature provides a suitable testbed for annotation.

6.1 GUM Ablations

For a controlled evaluation, our hypotheses require baselines for evaluation. Generating an accurate proposition can be trivial—GUMs could generate obvious and uninteresting facts about the user (e.g. `[Anon] is using their computer`). We therefore evaluate relative quality between generated propositions, using the following ablations as conditions:

- **ABLATION: No {Retrieve, Revise}**. In this ablation, we test a version of GUM that only *creates* propositions with no confidence. The GUM is unable to retrieve past context—generations are constructed given only the current email.
- **ABLATION: No {Retrieve}**. Here, we enable just the Revision model, revising old propositions based on current emails using a sliding window of past propositions (that fit into the model's context window) without the full retrieval engine.
- **FULL:** The full GUM architecture uses both the Revise and Retrieve modules. enabling retrieval over longer timespans and appropriate proposition revision.

6.2 Procedure

Our procedure involves a data loading process, where the participant exports the last 200 emails (threads, attachments, links, etc.) onto their personal computer. Then, participants train their own GUM, sequentially feeding inputs in the same order they appear in the inbox. Finally, we ask participants to label and rank outputs from the GUM and its ablations for quantitative evaluation, and share open-ended feedback on aspects of the study.

6.2.1 *Participants and Infrastructure.*

Participants. We recruited $N = 18$ participants through a mix of snowball sampling, word-of-mouth, and workplace channels at our institution. We required participants to be over 18 years old, be fluent in English, and have an active Gmail account. The median age was 26, with 8 identifying as male and 10 as female. 39% identified as Caucasian, 50% identified as Asian, and 5.5% as Hispanic, and 5.5% as other. Most of our participants were recruited from an academic institution: 14 of our participants have a bachelor's degree, 3 have a master's degree, and 1 has a GED.

Infrastructure. In serving GUM, we *only* use open-source models, relying on Llama 3.3 (70B parameters) for proposition and revision steps. We used quantized versions for inference speed and memory efficiency on 80GB NVIDIA H100 GPUs (which were provisioned by us directly and only accessible to the research team), using the Skypilot [77] library for autoscaling and load-balancing.

6.2.2 Procedure Details. Participants first exported 200 emails from their Gmail "Primary Inbox" (excluding Promotions and Social emails) via an export script that preserved read/unread states and replies. They then executed a training script that sequentially fed these emails into GUM in chronological order, with processing occurring on our private infrastructure while propositions and metadata were saved locally. Following training, each participant evaluated 30 stratified propositions across both time and confidence (ten from each condition) by providing binary accuracy judgments and ranking sets in pairwise comparisons with allowance for ties. Finally, participants completed a survey with open-ended questions (in Appendix C) about the strengths and weaknesses of the generated propositions, providing qualitative insights to complement the quantitative evaluations.

6.3 Evaluation Methods

To evaluate GUM, we assessed accuracy, calibration, and relative performance across ablations. We computed accuracy across the labeled propositions and compared results via t-tests. For relative rankings, we converted participant comparisons into pairwise win rates with confidence intervals, and applied significance testing using Holm-corrected binomial tests. To measure calibration, we calculated Brier scores (mean squared difference between predicted probabilities and actual outcomes), with lower scores indicating better calibration.

$$\text{Brier} = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2 \qquad (5)$$

Because GUM generates confidence scores on a $1 - 10$ scale, we normalize these scores to the [0,1] range before computing the Brier score. Finally, we conducted t-tests for Brier across GUM ablations.

6.4 Results

We outline a sample of anonymized and correctly annotated propositions from our evaluation below to illustrate the breadth of inferences constructed from email alone.

```
[ANON] values their privacy and is interested
in how their personal information is used.
```
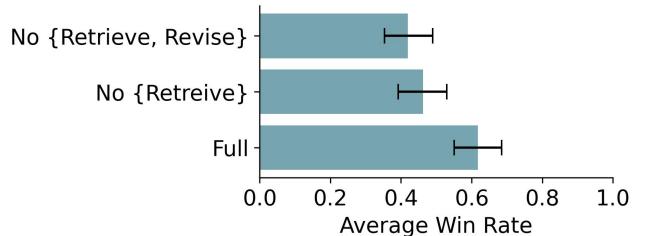

Figure 5: Relative to ablations, propositions from the full GUM architecture are preferred by users. Additionally, retrieval and revision depend heavily on each other. Without retrieving relevant propositions, the revise module is unable to correctly adjust propositions. Ablating retrieval causes win rates to fall significantly.

Variants	Accuracy	Calibration (Brier)
No {Retrieve, Revise}	74.41 ± 7.81	–
No {Retrieve}	73.28 ± 4.42	0.36 ± 0.07
Full	76.15 ± 5.85	0.17 ± 0.03

Table 1: In isolation, accuracy for GUM and its ablations is about the same—models generally construct correct inferences about the user. However, the full GUM architecture produces propositions that are significantly more calibrated.
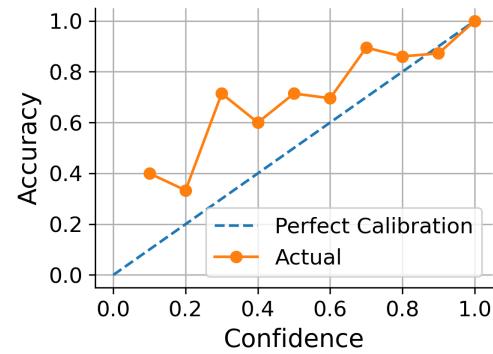

Figure 6: GUMs are generally well calibrated. When errors occur, GUMs are *underconfident* in their propositions—the actual model's predictions lie above perfect calibration. In the user modeling setting, this is ideal. We should underestimate propositions to avoid eroding user trust.

```
[ANON] is likely a frequent traveler, given
their engagement with emails about airline
transfer bonuses and credit card rewards.

[ANON] is annoyed by notifications from
Academia.edu.
```

```
[ANON] is proactive about managing their
notifications and email subscriptions.

[ANON] is aware of the risks associated
with investing, including the possible loss
of money.
```

The full architecture bests GUM ablations. To differentiate between ablations and to identify relative performance of the full GUM, we asked participants to *rank* ablations. In the ranked setting, the difference between the full GUM the ablations are clear (Fig 5). Win rates for the full model are higher than the ablations (μ, 95% conf; FULL = 0.618 ± 0.07 , No {Retrieve} = 0.463 ± 0.07, No {Retrieve, Revise} = 0.420 ± 0.07). This difference is also validated by a bootstrapped significance test following the Holm-Beniforri correction ($p < 0.05$). Between the No {Retrieve, Revise} and No {Revise} ablations, however, we see no significant difference. We suspect that relevant email is often spaced apart, and that revision *without* retrieval (e.g. the sliding window approach) fails to capture long-range dependencies between emails. Without this, Revision falls apart entirely.

Propositions generated by the user model are both calibrated and accurate. On average, participants rank GUM propositions as correct 76.15 ± 5.85 of the time. Participants were also fairly positive on the general accuracy of propositions.

> "People should use this instead of asking for zodiac signs or MBTIs."

On average, participants also rated propositions from email alone as accurate to very accurate (μ = 6.47 on a 1-7 Likert scale). In addition, we find that GUMs are well calibrated, with a Brier score of 0.17 ± 0.03. Almost all of GUMs' calibration error comes from being *underconfident* (see Fig 6). In our setting, we'd prefer that GUMs are underconfident rather than overconfident: an overconfident user model might wrongly assume it knows a user's preferences, while underconfidence allows GUM to adapt more cautiously. However, generating an accurate proposition alone is too trivial a task. All GUM ablations, for example, generate equally accurate propositions (μ, 95% conf.; FULL = 76.15 ± 5.85, No {Retrieve} = 73.28 ± 4.42, No {Retrieve, Revise} = 74.41 ± 7.81)). No condition was significantly stronger than another (see Table 1).

"It feels kind of weird to see this in writing." Propositions generated by the GUM are somewhat blunt. One participant occasionally used delivery services to order food. From the sequence of emails delivered to the participant's inbox, GUM concluded that the participant `valued convenience and was willing to pay for it`. For another participant, GUM correctly inferred that the user `may not always have the time to read all their emails immediately`. Propositions that reveal uncomfortably accurate insights often have lower confidence scores. Still:

> "It is not wrong per se, but it feels kind of weird seeing this in writing."

Emails are one-dimensional proxy for true user context. Despite the overall accuracy of GUM's propositions, a few participants noted that emails were still a highly constrained representation of their context:

> "The emails I get on this account are not representative of my life."

Errors in propositions often emerged from this fact. Some GUMs would over-index on a particular facet of a participant's life, focusing mostly on financial decisions, a particular hobby, or a recent purchase. In our end-to-end evaluation to come (§8), we expand the GUM's viewpoint substantially, feeding it screenshots from participants' computers over the course of 5 days.

7 Evaluating General User Models: Privacy Audit Module

As we expand context for GUM, auditing observations for privacy becomes increasingly critical (§5.2). During an Audit, the GUM uses itself to make inferences about an individual's contextual privacy preferences when processing interaction data. We explicitly test this, hypothesizing that GUM helps when making inferences related to a user's privacy preferences. We evaluate the Audit module using the same email setting from our Accuracy and Calibration eval (§6), with the same set of $N = 18$ participants (§6.2.1).

7.1 Procedure

We asked participants to skip annotating any propositions that violated their personal privacy preferences, effectively flagging them in the process. Using this signal, we computed the number of flagged propositions overall. Participants were also presented with the GUM's responses to Nissenbaum [50]'s contextual integrity questions—these questions were used to power the Audit module in §5.2. Participants were asked to select the best generation across GUM ablations and score the accuracy of the best *Audit* response using a 1-7 Likert score. As reference, the author's anonymized contextual integrity generation is below:

> Based on the emails, you have shared your:
>
> 1. Email address with Calendly, Nextdoor, OpenAI, Instacart, and GitHub.
> 2. GitHub account with [ANON], who has invited you to collaborate on a repository...
>
> You have shared this data in the following contexts:
>
> 1. Scheduling events and meetings (Calendly)
> 2. Receiving recommendations and updates from your community (Nextdoor) ...
>
> You have not responded to or engaged with the Nextdoor email, which could suggest that you are not interested in sharing data or interacting with that platform.

7.2 Results

GUMs generally respect contextual integrity. From the 180 propositions annotated by participants for the full GUM, we found only 7 propositions that were flagged as contextual integrity violations. On average, participants agreed or strongly agreed with responses to contextual integrity generations (μ = 6.06 on a 1-7 Likert scale).

Furthermore, a majority of participants 61.1% selected the full GUM as generating the best contextual integrity response, compared to ablations without retrieval (22.2%) and revision (16.6%). As a sanity check, we confirmed that no propositions included verbatim references to credit card numbers, phone numbers, or addresses.

But when they don't, it's bad. Seven observations *still* bypassed the Audit step—a non-negligible number. We looked into each participant's open-ended responses for insight. One participant (two violations) mentioned some surprise at the GUM being able to recall specific names from their email:

> "The ones which brought up messages or social media with other people and then explicitly named some of my friends were a little surprising, though I can imagine where all that information is sourced from within my emails."

The broader conclusion: GUMs are able to extract a surprising number of inferences from just 200 emails. For some users, we suspect that making explicit social inferences about others is a boundary for GUM. In principle, the user can explicitly dictate their desired contextual privacy norms to their GUM, but there is a delicate trust balance to walk. We revisit this in our discussion.

Another participant (with 3 violations) left the following response:

> "[The proposition was] based on a phishing email I received rather than my actual email, and it had a decently high confidence rating, but it wasn't accurate at all, and so the model thought it was me and my interest rather than just a phishing attempt."

GUMs can be maliciously re-written by advertisements and spam, opening them up to attackers. Prompt-injection attacks are a well-known VLM vulnerability [83]; *however*, GUM exposes a larger surface area to exploit this attack. Just like prompt injection attacks can be mitigated by red-teaming a model, we believe that GUM injection attacks can be mitigated—though not entirely erased without further research.

8 Evaluating General User Models: End-to-End via GUMBO

In our final evaluation, we assess the overall effectiveness of GUM through a formative study with GUMBO, a system that constructs a GUM through screenshots of the user's screen and marshalls it to generate suggestions.

8.1 Procedure

Our study consists of two main parts: a **burn in**, where participants configure GUMBO on their computer and have it silently learn and construct their GUM; and **active use**, where participants interact with GUMBO suggestions. In this section, we outline the infrastructure for serving GUM and detail each evaluation stage. Our study procedure and participant recruitment was approved by our institution's IRB (up to 5 participants to mitigate any risks associated with GUMBO errors in a larger sample).

8.1.1 Participants and Infrastructure. We use the same infrastructure in our technical evaluation, except that we deploy a vision

model (Qwen 2.5 VL 72B [7]) for the Screen Observer, transcribing observations from the screen. Participants were recruited using a mix of snowball sampling and word-of-mouth. Given the level of trust required to use a system like GUMBO, participants were either acquaintances of the authors or acquaintances of acquaintances. No participants had any knowledge of the system before the study. Participants were located in the United States and were over 18 years of age. 2 participant identified as White, 2 as Asian, and 1 as African American, and all had bachelor's degrees. Participation required about an hour for synchronous onboarding and offboarding; compensation was $75 USD.

8.1.2 Procedure Details. Participants began the study with onboarding, installing GUMBO, optionally providing personal details, and completing a features tutorial. The subsequent **burn in** phase lasted 24 hours, during which GUMBO collected data without showing suggestions, addressing the cold-start problem. The **active use** phase then spanned four days, with participants actively engaging with suggestions. During offboarding, we collected annotations and feedback on suggestions. We additionally conducted semi-structured interviews discussing participants' propositions, experiences, and privacy concerns.

8.2 Analysis

We first compute calibration and accuracy across all participants' labeled propositions, just like in our initial technical evaluation. In addition, the first author then conducted two rounds of qualitative open-coding across the transcripts. In the first round, the author labeled low-level themes across transcripts. In the second round, common themes were aggregated.

8.3 Results

We synthesize findings across both a quantitative analysis of GUM accuracy and qualitative themes from our interviews.

GUMs remain accurate and calibrated in our end-to-end evaluation. Mirroring results from our email evaluation, participants found sampled propositions to be accurate (0.79 ± 0.07) and calibrated (Brier = 0.28 ± 0.04), with error coming from underconfident predictions. Propositions that reflected factual statements about the user were often annotated or discussed with very little hesitation.

> P1 is conducting research on health and behavior interventions.
>
> P3 works on research related to civil and environmental engineering.
>
> P4 is an employee at [ANON] company.

Participants also generally observed that these propositions were of higher confidence. For these propositions, participants expressed little surprise.

> "What's so special about this? It's just me." - P4

In interviews, participants were quick to rationalize why these propositions were true, citing concrete websites, files, or messaging exchanges that explained how these propositions were generated.

Maybe too accurate... When reviewing propositions, participants were conflicted about generations that passed a judgement on values, skills, or norms. Consider the following propositions (all labeled correct by participants):

> P1 is struggling with fixing bugs in their research project.
>
> P2 may be experiencing stress or pressure due to a large number of unread emails and notifications.
>
> P3 prioritizes communicating with their advisor over other people.
>
> P4 is unhappy with their job.

Here, participants expressed a range of different opinions. P4 and P5 found these propositions both unserious and entertaining, ascribing little weight to them. P1, on the other hand, found them judgmental. In contrast, P2 immediately began self-reflecting:

> "I felt so validated by that [proposition on stress]. I totally feel pressure when I get another email or notification. It's validating to see it notice this." - P2

While P3 reflected on why they prioritized communication with their advisor,[6] they mentioned that only a subset of propositions were useful for reflection—propositions related to their abilities were not particularly constructive for them.

> "I'd hide [propositions on ability], honestly. Even if they were true! Just show me the suggestions." - P3

It was also trickier for participants to rationalize why their GUM was making these inferences. For participants who enjoyed the self-reflection parts of GUM, coming to their own conclusion was a boon; for others, this lack of transparency was annoying. We revisit self-reflection in the discussion (§9.1).

GUMBO provides strong-to-excellent suggestions for all participants. All participants came to interviews especially excited about a suggestion generated by GUMBO. On our 7 point Likert scale, 25% of suggestions were ranked as strong (6) or excellent (7). P2 and P3 wanted to keep the system running on their computer after the study concluded (with some modifications to proposition visibility), while P2 and P5 took screenshots of a handful of suggestions to keep before offboarding. Suggestions that were rated highly often made extensive use of the underlying GUM.

> "It knew who my roommate was; what our budget was; where we were moving; that I was worried about this move. It worked backward from our move-in date, planned a schedule, and identified moving services. And half of my conversation with my roommate wasn't in English." - P1

Overall, we found that useful suggestions generally fell into two main categories: immediate, low-level assistance, and opportunities where participants hadn't realized an AI model could be helpful.

In the moment help. A subset of helpful suggestions were low-level, helping the user with what they were currently doing. P5, for example, was in the process of transferring email inboxes from their institution's email to Gmail, and never figured out how to enable

---

[6]The author of this paper also reflected on this.

notification on Gmail. GUMBO generated a step-by-step guide—P5 was thrilled. In a similar flavor, P3 was designing a presentation for an upcoming quals exam. GUMBO provided tips on slide transitions which P3 first tried and then adopted.

"I didn't even think about that!" A more exciting class of suggestions generated by GUMBO addresses needs that aren't explicitly signaled by the user. GUMBO proactively generated a brainstorming outline for a framework P3 was considering using in their research, using tools P3 was comfortable with:

> "What I typically like to do when I run into a new idea is create a barebones Overleaf document and outline [how the idea is] related to my work. I did not even tell this system anything, but it identified that I have this habit. And it created this entire outline—in LateX—of how I could write my paper in the context of this new framework I was checking out. I was like, wow." - P3

P5 spent some time watching Twitch streams of games. For P5, GUMBO identified specific characters of interest based on where and how long they paused the stream; and how often they replayed specific portions. Using this information, GUMBO identified and surfaced new streamers that also played with the same character. P5 ended up exploring and watching content from GUMBO's recommended streamers.

Contextual agency. Participants raised agency as a concern when interacting with GUM, in terms of being able to control which contexts observations were turned into propositions or suggestions. P3 suggested that the system clarify with users when generating low-confidence propositions from observation, but acknowledged that the assumed propositions eventually did lead to useful suggestions.

> "Maybe I wasn't familiar with PowerPoint, but if it asked and then used that to give me a suggestion I think I would've been happier with it." - P3

For suggestions, GUMBO did not replace what the user liked. P4, for example, really enjoys travel planning themselves, and their underlying GUM included this fact. When generating suggestions, GUMBO generated only ideas for travel instead of generating a full travel plan (something the system was capable of doing):

> "I like that it gave me some ideas for adjusting my honeymoon schedule. It didn't try to redo the whole thing or make one from scratch." - P4

8.4 Errors and Boundaries

Privacy, Privacy, and Privacy. Privacy guarantees were critical from the start. The two participants that were acquainted with the author explicitly mentioned this bias (which we revisit in the limitations). Recruitment alone required trust in the designer and that participants' GUMs ran on self-hosted models.

> "If I didn't know you and trust you, I would never install this thing." - P4

Still, some participants (P1, P2) habituated and occasionally forgot that GUMBO was even processing interaction data.

"It was intimidating at first. And I was also like shit like, do I need to be careful what I say? And then after a day I was like, whatever F it." - P2

Others were keenly aware of the GUMBO's presence during the entire duration of the study (P3).

"It was just identifying things about me and understanding how I work, my work in general, and that was actually kind of scary." - P3

Only after returning to the list of propositions would participants realize that GUMBO was still active. On the other hand, P5 expressed no reservations during the entire study.

From General User Models to General Clippy Models? All participants felt like GUMBO often tried to recommend and execute on suggestions it was incapable of doing, a practical limit with agentic AI execution given today's models [8, 66]. A good subset of suggestions (20.69%) were labeled as on-track, but failed during execution. Still, some participants exacted joy from watching GUMBO try and fail to execute.

"So my advisor's been asking me to listen to this Claude Steele podcast—it's been stuck at the top of my to-do list forever. I just never got around to it. Then this system gave me a nudge. It looked into the podcast and said it drew connections to my work. The outline was decent, but the connections were total garbage. Still, it got me to finally listen, and I ended up totally locked in, working on it for hours after. I almost prefer this, because it didn't take away any cognitive burden." - P2

Still, this failure mode with GUMBO reflects capability or tool-use limitations of the underlying LLM: GUMBO did not explore the internet and ingest the podcast. Even simpler suggestions, such as scheduling reminders, also fail. GUMBO claimed to have "scheduled reminders" for P5, allowing him to discreetly watch an NCAA game during his work hours. However, GUMBO had no access to notification APIs, so execution failed entirely.

Large ~~Language Models~~ Eager Beavers. Eagerness to help was a recurring theme amongst poorer suggestions. GUMBO would jump the gun on generating suggestions while the user was in the process of finishing the suggestion itself. Or GUMBO would try to cheat by suggesting something the participant had already completed. There's a fine line between suggestions that are in-the-moment helpful and suggestions that are too late.

Inferences on Sensitive Topics. P1 expressed some discomfort with GUMBO making and saving inferences on sensitive topics. Here, GUMs audit module incorrectly assumed that because P1 viewed and discussed topics with others, it was O.K. for it to save this information too. P1 mentioned that the integrity standards for GUM should be strictly stronger than what their context implies—and that the Audit module should itself be auditable.

(Is) the screenome limited? Does it generalize? All participants reflected on the generalizability of a GUM trained on their computer use, and highlighted how events outside of their computer would not be captured by GUMBO.

"There were times that I was doing something on my phone and I was like, oh, I wish I could capture this too." - P2

P3 also raised a similar point, explicitly reflecting on the privacy-capability trade-off of adding more context to a system like GUMBO.

"There were some really great things that came out of there that I wouldn't have thought about. So how do I amplify that, but mitigate [seeing judgmental propositions]. I think I would need more than five days to use it. I would integrate some more personal work that I do." - P3

We revisit this paradox in the discussion (§9.2).

9 Discussion

Across our technical evaluations and end-to-end deployment, we find that GUM shows promise in various human-computer interaction scenarios. In our discussion, we focus on the implications of deploying GUMs. We revisit how participants engaged and reflected with propositions, the GUM privacy paradox, and the ethical and societal implications of widely deployed GUMs.

9.1 Reflecting on Propositions

We never intended or designed GUMBO to serve as a self-reflection tool for participants. However, across both our email and end-to-end evaluation, many participants engaged deeply with the content of the propositions. For some participants, GUM propositions were too candid: they explicitly constructed value judgements and made assessments about a user's priorities. This could be a side effect of our prompt—we explicitly prompt GUM to make accurate and truthful assessments about the user. And these assessments are indeed useful for downstream suggestions generated by GUMBO. Without an accurate user model, via an honest assessments of where the user might need help, downstream applications might fail to provide assistance. However, an accurate GUM might enable a user to indulge in activities that, while they think are helpful, might not truly be helpful (e.g. cancelling advising meetings to make time for making/eating more ice cream).[7]

This dichotomy—between the ideal self and true self—is a well documented psychological phenomena [32, 59]. When confronted with their GUM, participants' reactions ranged from feeling judged to engaging in productive self-reflection.[8] Our work raises key design questions: should GUMBO provide raw access to underlying user models, present sanitized versions, or hide propositions entirely? One approach might adapt the visible representation based on contextual factors, showing reflective content only when users are receptive to it [26]—similar to how our Audit module uses GUM to make filtering decisions for privacy.

9.2 The Privacy Paradox

The privacy paradox observes that people often say they care very much about their online privacy, but willingly give it up in practice in exchange for a benefit [54]. For most applications, this means

---

[7]This is a point of contention between the authors. Some of the authors are productive (the advisors), and would benefit greatly from an accurate user model. The first author thinks the "Real Them" would enjoy procrastinating.

[8]More than once, participants likened GUM to a BuzzFeed personality test.

giving up sensitive personal information such as location data or browsing habits to targeted advertisers via social media accounts. However, the GUM is a much more general and, we argue, much more powerful user model. So, privacy behavior becomes a critical question.

The more a GUM sees, the more precise it becomes, and the stronger its downstream recommendations. Users can improve a GUM by giving it anything that the model is capable of parsing. While several of the participants were initially focused on privacy risks, we observed that they naturally came to the same conclusion and began voluntarily offering up additional data to the GUM. In our end-to-end evaluation, two of our five participants wanted to continue using a version of GUMBO following the study, brainstorming ways to give GUM more data *despite* initial privacy concerns.

If one believes that sharing data with a private, local GUM is safe and desirable, then disclosure becomes self-reinforcing: as users observe concrete improvements in the model's performance proportional to their data contributions, they may share more. However, if one considers the accidental disclosure risks or security risks of GUMs, this loop could backfire. With this context, we outline a range of ethical and societal considerations in using GUMs.

9.3 Ethical and Societal Risks

Persuasion, advertising, and surveillance. One risk we foresee lies in using GUMs to target users, either through persuasion, advertisement, or surveillance. *We strongly recommend that all GUMs are trained and hosted on the end-users' computer or on personal infrastructure.* This is a guiding principle throughout our work—all models used to build a GUM are open source, and infrastructure is managed by the research team. If external systems are ever granted access to the GUM, access should be controlled and audited. We also foresee risks associated with third-parties attempting to train GUMs, without consent, on a user's activity within their application. In these instances, obfuscation systems can generate interaction data to obscure genuine user behavior [51].

Manipulating GUMs. In our email evaluations, we found evidence of unintentional GUM manipulation. Spam email, ingested by the GUM, maliciously edited a GUM's underlying propositions. Instead of targeting users, attackers could craft messages that manipulate a user's GUM. Any application that relies on a GUM would therefore be affected. One possible mitigation for this risk would be for the GUM to ask the user to confirm any inputs that seem out of character based on its understanding so far. Another would be to develop the equivalent of adblock or spam filtering techniques [60]) to proactively detect and intercept maliciously crafted content before it's processed by GUM, or the model would need to include a component that reflects on whether someone is trying to trick it.

Bias. While participants did not raise concerns related to biased inferences, GUMs build on biased models [22, 73]. Analogously, recommendation systems have a long history of surfacing biased recommendations to end users [52]. GUMs effectively expand the surface area of recommendation systems. Carefully red-teaming and understanding the types of inferences constructed by GUM across a larger set of adversarially constructed contexts is necessary before deployment.

9.4 Limitations and Future Work

Sampling bias. A challenge in recruiting participants for our end-to-end evaluation lies in trust. To this end, our recruitment occurred primarily at our institution and through word-of-mouth / snowball sampling. Our institution's IRB permitted up to 5 participants to mitigate the impact of any potential issues with GUMBO. For now, we recommend considering our results as likely generalizing most strongly to technical users who have familiarity with AI.

Hallucinations and misattributions. Hallucinations remain an issue with current large models. While incorrect propositions are indeed appropriately calibrated, a handful are still confident and incorrect. GUM would also occasionally hallucinate applications on a user's computer that didn't exist, or connect two unrelated propositions.

The screen is still a narrow proxy for context. While we see a wide range of the user via the GUM attached to a user's email or their screen, it's not a complete picture—GUMs are still far from building context through everyday action [23]. If a user does something outside the scope of the GUM, it will fail to generalize. However:

Better models will extend beyond vision and language. The current GUM implementation relies on vision and language models, constraining the input space. Large models that ingest a wider range of modalities [9] and diverse tool uses could enhance GUM's capabilities. Already, advanced speech models can discern a user's tone, while healthcare models interpret various sensor signals—all of which in principle could integrate into the GUM.

10 Conclusion

We introduce GUMs, computational models that learn rich representations of user context by observing everyday behavior. GUMs transform unstructured interactions into confidence-weighted propositions, continuously refining their inferences as new evidence accumulates. GUMs open up a range of possibilities, from grounding language models in observed user behavior to enabling proactive systems that can act on a user's behalf. We demonstrate the utility of GUMs through GUMBO, an interactive assistant that surfaces helpful suggestions drawn directly from personal context and attempts to complete them. In evaluations, GUMs produce rapid, calibrated, and accurate inferences about users. We argue that GUMs provide a framework for moving beyond siloed application-specific user models toward a *general* user model—one that understands who you are across the many contexts of your life.

References

[1] Gavin Abercrombie, Amanda Cercas Curry, Tanvi Dinkar, and Zeerak Talat. 2023. Mirages: On Anthropomorphism in Dialogue Systems. *arXiv preprint arXiv:2305.09800* (2023).

[2] Maneesh Agrawala. 2023. Unpredictable black boxes are terrible interfaces.

[3] John R. Anderson. 1993. *Rules of the Mind.* Lawrence Erlbaum Associates, Hillsdale, NJ.

[4] Chinmaya Andukuri, Jan-Philipp Fränken, Tobias Gerstenberg, and Noah D Goodman. 2024. Star-gate: Teaching language models to ask clarifying questions. *arXiv preprint arXiv:2403.19154* (2024).

[5] Apple Computer. 1987. Knowledge Navigator. Conceptual video by Apple illustrating a vision for the future of personal computing..

[6] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L Glassman. 2024. Chainforge: A visual toolkit for prompt engineering and llm hypothesis testing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems.* 1–18.

[7] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923* (2025).

[8] Gagan Bansal, Jennifer Wortman Vaughan, Saleema Amershi, Eric Horvitz, Adam Fourney, Hussein Mozannar, Victor Dibia, and Daniel S Weld. 2024. Challenges in Human-Agent Communication. *arXiv preprint arXiv:2412.10380* (2024).

[9] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[10] Susan E Brennan. 2014. The grounding problem in conversations with and through computers. In *Social and cognitive approaches to interpersonal communication*. Psychology Press, 201–225.

[11] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. 335–336.

[12] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. 2025. Why Do Multi-Agent LLM Systems Fail? *arXiv preprint arXiv:2503.13657* (2025).

[13] Maximillian Chen, Xiao Yu, Weiyan Shi, Urvi Awasthi, and Zhou Yu. 2023. Controllable Mixed-Initiative Dialogue Generation through Prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 951–966. doi:10.18653/v1/2023.acl-short.82

[14] Herbert H Clark. 1996. *Using language*. Cambridge university press.

[15] Herbert H Clark and Susan E Brennan. 1991. Grounding in communication. (1991).

[16] Herbert H Clark and Edward F Schaefer. 1989. Contributing to discourse. *Cognitive science* 13, 2 (1989), 259–294.

[17] Alan Demers, Srinivasan Keshav, and Scott Shenker. 1989. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer communication review* 19, 4 (1989), 1–12.

[18] Anind K Dey. 2001. Understanding and using context. *Personal and ubiquitous computing* 5 (2001), 4–7.

[19] Anind K Dey and Gregory D Abowd. 2000. Cybreminder: A context-aware system for supporting reminders. In *Handheld and Ubiquitous Computing: Second International Symposium, HUC 2000 Bristol, UK, September 25–27, 2000 Proceedings 2*. Springer, 172–186.

[20] Anind K Dey, Gregory D Abowd, and Daniel Salber. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human–Computer Interaction* 16, 2-4 (2001), 97–166.

[21] Anind K Dey, Daniel Salber, Gregory D Abowd, and Masayasu Futakawa. 1999. The conference assistant: Combining context-awareness with wearable computing. In *Digest of Papers. Third International Symposium on Wearable Computers*. IEEE, 21–28.

[22] Emily Dinan, Angela Fan, Ledell Wu, Jason Weston, Douwe Kiela, and Adina Williams. 2020. Multi-dimensional gender bias classification. *arXiv preprint arXiv:2005.00614* (2020).

[23] Paul Dourish. 2004. What we talk about when we talk about context. *Personal and ubiquitous computing* 8 (2004), 19–30.

[24] Anton N Dragunov, Thomas G Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L Herlocker. 2005. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 10th international conference on Intelligent user interfaces*. 75–82.

[25] Gerhard Fischer. 2001. User modeling in human–computer interaction. *User modeling and user-adapted interaction* 11 (2001), 65–86.

[26] Rowanne Fleck and Geraldine Fitzpatrick. 2010. Reflecting on reflection: framing a design landscape. In *Proceedings of the 22nd conference of the computer-human interaction special interest group of australia on computer-human interaction*. 216–223.

[27] James Fogarty, Scott E Hudson, Christopher G Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C Lee, and Jie Yang. 2005. Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)* 12, 1 (2005), 119–146.

[28] James Fogarty, Amy J Ko, Htet Htet Aung, Elspeth Golden, Karen P Tang, and Scott E Hudson. 2005. Examining task engagement in sensor-based statistical models of human interruptibility. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 331–340.

[29] Yujian Gan, Changling Li, Jinxia Xie, Luou Wen, Matthew Purver, and Massimo Poesio. 2024. ClarQ-LLM: A Benchmark for Models Clarifying and Requesting Information in Task-Oriented Dialog. *arXiv preprint arXiv:2409.06097* (2024).

[30] Yifan Gao, Henghui Zhu, Patrick Ng, Cicero dos Santos, Zhiguo Wang, Feng Nan, Dejiao Zhang, Ramesh Nallapati, Andrew O Arnold, and Bing Xiang. 2021. Answering Ambiguous Questions through Generative Evidence Fusion and Round-Trip Prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 3263–3276.

[31] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[32] E Tory Higgins. 1987. Self-discrepancy: a theory relating self and affect. *Psychological review* 94, 3 (1987), 319.

[33] Jihyeong Hong, Yokyung Lee, Dae Hyun Kim, DaEun Choi, Yeo-Jin Yoon, Gyucheol Lee, Zucheul Lee, and Juho Kim. 2024. A Context-Aware Onboarding Agent for Metaverse Powered by Large Language Models. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 1857–1874.

[34] Joey Hong, Sergey Levine, and Anca Dragan. 2023. Zero-Shot Goal-Directed Dialogue via RL on Imagined Conversations. *arXiv preprint arXiv:2311.05584* (2023).

[35] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.

[36] Eric Horvitz. 2006. Machine learning, reasoning, and intelligence in daily life: Directions and challenges. In *Proceedings of*, Vol. 360.

[37] Eric Horvitz, Susan Dumais, and Paul Koch. 2004. Learning predictive models of memory landmarks. In *Proceedings of the annual meeting of the cognitive science society*, Vol. 26.

[38] Eric J Horvitz, John S Breese, David Heckerman, David Hovel, and Koos Rommelse. 2013. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. *arXiv preprint arXiv:1301.7385* (2013).

[39] Scott Hudson, James Fogarty, Christopher Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny Lee, and Jie Yang. 2003. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 257–264.

[40] Anthony Jameson. 2001. Modelling both the context and the user. *Personal and Ubiquitous Computing* 5 (2001), 29–33.

[41] Peiling Jiang, Jude Rayan, Steven P Dow, and Haijun Xia. 2023. Graphologue: Exploring large language model responses with interactive diagrams. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*. 1–20.

[42] Matthew Jörke, Shardul Sapkota, Lyndsea Warkenthien, Niklas Vainio, Paul Schmiedmayer, Emma Brunskill, and James Landay. 2024. Supporting physical activity behavior change with llm-based conversational agents. *arXiv preprint arXiv:2405.06061* (2024).

[43] Chaitanya K Joshi, Fei Mi, and Boi Faltings. 2017. Personalization in goal-oriented dialog. *arXiv preprint arXiv:1706.07503* (2017).

[44] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2022. CLAM: Selective Clarification for Ambiguous Questions with Large Language Models. *arXiv preprint arXiv:2212.07769* (2022).

[45] Daniel Heiskell Lassiter. 2012. Presuppositions, provisos, and probability. *Semantics and Pragmatics* (2012), 1–37.

[46] JM Lucas, F Fernández, J Salazar, Javier Ferreiros, and Rubén San Segundo. 2009. Managing speaker identity and user profiles in a spoken dialogue system. *Procesamiento del lenguaje natural* 43 (2009), 77–84.

[47] Pattie Maes and Robyn Kozierok. 1993. Learning interface agents. In *AAAI*, Vol. 93. 459–465.

[48] Sabrina J Mielke, Arthur Szlam, Emily Dinan, and Y-Lan Boureau. 2022. Reducing conversational agents' overconfidence through linguistic calibration. *Transactions of the Association for Computational Linguistics* 10 (2022), 857–872.

[49] Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering Ambiguous Open-domain Questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 5783–5797.

[50] Helen Nissenbaum. 2004. Privacy as contextual integrity. *Wash. L. Rev.* 79 (2004), 119.

[51] Helen Nissenbaum and Howe Daniel. 2009. TrackMeNot: Resisting surveillance in web search. (2009).

[52] Safiya Umoja Noble. 2018. Algorithms of oppression: How search engines reinforce racism. In *Algorithms of oppression*. New York university press.

[53] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[54] Patricia A Norberg, Daniel R Horne, and David A Horne. 2007. The privacy paradox: Personal information disclosure intentions versus behaviors. *Journal of consumer affairs* 41, 1 (2007), 100–126.

[55] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. (2021).

[56] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. (2023).

[57] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra

of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*. 1–22.

[58] Meredith Ringel, Ed Cutrell, Susan Dumais, and Eric Horvitz. 2003. Milestones in time: The value of landmarks in retrieving information from personal stores. In *Proceedings of Interact 2003*.

[59] Carl Ransom Rogers. 1995. *On becoming a person: A therapist's view of psychotherapy*. Houghton Mifflin Harcourt.

[60] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. 1998. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, Vol. 62. Citeseer, 98–105.

[61] Bill Schilit, Norman Adams, and Roy Want. 1994. Context-aware computing applications. In *1994 first workshop on mobile computing systems and applications*. IEEE, 85–90.

[62] Omar Shaikh, Kristina Gligorić, Ashna Khetan, Matthias Gerstgrasser, Diyi Yang, and Dan Jurafsky. 2023. Grounding gaps in language model generations. *arXiv preprint arXiv:2311.09144* (2023).

[63] Omar Shaikh, Michelle S Lam, Joey Hejna, Yijia Shao, Hyundong Justin Cho, Michael S Bernstein, and Diyi Yang. [n. d.]. Aligning Language Models with Demonstrated Feedback. In *The Thirteenth International Conference on Learning Representations*.

[64] Omar Shaikh, Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2025. Navigating Rifts in Human-LLM Grounding: Study and Benchmark. arXiv:2503.13975 [cs.CL] https://arxiv.org/abs/2503.13975

[65] Shreya Shankar, JD Zamfirescu-Pereira, Björn Hartmann, Aditya Parameswaran, and Ian Arawjo. 2024. Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[66] Yijia Shao, Vinay Samuel, Yucheng Jiang, John Yang, and Diyi Yang. 2024. Collaborative Gym: A Framework for Enabling and Evaluating Human-Agent Collaboration. *arXiv preprint arXiv:2412.15701* (2024).

[67] Ben Shneiderman and Pattie Maes. 1997. Direct manipulation vs. interface agents. *interactions* 4, 6 (1997), 42–61.

[68] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In *Proceedings of the 36th annual ACM symposium on user interface software and technology*. 1–18.

[69] Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher D Manning. 2023. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models finetuned with human feedback. *arXiv preprint arXiv:2305.14975* (2023).

[70] Polina Tsvilodub, Michael Franke, Robert D Hawkins, and Noah D Goodman. 2023. Overinformative Question Answering by Humans and Machines. *arXiv preprint arXiv:2305.07151* (2023).

[71] Priyan Vaithilingam, Ian Arawjo, and Elena L Glassman. 2024. Imagining a future of designing with ai: Dynamic grounding, constructive negotiation, and sustainable motivation. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference*. 289–300.

[72] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[73] Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. 2021. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359* (2021).

[74] M Weiser. 1991. The Computer for the 21st Century. *Scientific American Ubicomp Paper after Sci Am editing* (1991).

[75] Mark Weiser and John Seely Brown. 1996. Designing calm technology. *PowerGrid Journal* 1, 1 (1996), 75–85.

[76] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080* (2021).

[77] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, et al. 2023. {SkyPilot}: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 437–455.

[78] JD Zamfirescu-Pereira, Heather Wei, Amy Xiao, Kitty Gu, Grace Jung, Matthew G Lee, Bjoern Hartmann, and Qian Yang. 2023. Herding AI cats: Lessons from designing a chatbot by prompting GPT-3. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*. 2206–2220.

[79] J Diego Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–21.

[80] Liyi Zhang, Michael Y Li, and Thomas L Griffiths. 2024. What should embeddings embed? autoregressive models represent latent generating distributions. *arXiv preprint arXiv:2406.03707* (2024).

[81] Michael JQ Zhang and Eunsol Choi. 2023. Clarify when necessary: Resolving ambiguity through interaction with lms. *arXiv preprint arXiv:2311.09469* (2023).

[82] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? *arXiv preprint arXiv:1801.07243* (2018).

[83] Yanzhe Zhang, Tao Yu, and Diyi Yang. 2024. Attacking Vision-Language Computer Agents via Pop-ups. *arXiv preprint arXiv:2411.02391* (2024).

A Code

A demo of Gumbo and our open source package for GUM are available at https://generalusermodels.github.io

B Abridged Prompts

Here, we outline prompts from our work in more detail. We discuss prompts used in the underlying GUM and the instantiated system, Gumbo.

B.1 GUM prompts

B.1.1 Calibration. GUM is responsible for generating confidence scores alongside each proposition. One method to do this is to directly look at the logprobs on the completion from the LLM. However, this estimate for instruction-tuned LLMs is often miscalibrated: LLMs are more confident than they really are. An alternative approach involves prompting LLMs for a verbalized confidence score [69]. This entire process occurs during proposition generation, after we've already generated the reasoning and the proposition:

```
observation: [screenshots of the user switching
between Overleaf and YouTube]

proposition_reasoning: "The user appears
distracted, switching focus between an ice
cream recipe video and typing intermittently
in an Overleaf window."

proposition: "User periodically views ice
cream recipes while writing."
```

Conditioned on the above, we elicit a support score (1-10), which serves as our confidence measure. An abridged version of our prompt is below:

```
Generate a support score that captures
how much evidence you have to support the
generated propositons. Be conservative in
your support estimates. Just because an
application appears on the screen does
not mean they have deeply engaged with
it. They may have only glanced at it for a
second, making it difficult to draw strong
conclusions.

Assign high support scores (e.g., 8-10)
only when the transcriptions provide explicit,
direct evidence that the user is actively
engaging with the content in a meaningful
way.
```

Using the above, we generate a support score:

```
support: 10
```

We elicit conservative confidence scores, as instructed in our prompt. GUM is overall well-calibrated and does indeed generate conservative support scores. We suspect that GUM's calibration could be improved either by finetuning or through better prompting—we leave this for future work.

B.1.2 Reranker. To marshall the GUM, we need to be able to retrieve relevant propositions. A challenge here requires *re-ranking* outputs—many of the retrieved propositions may be irrelevant. To

this end, we classify retrieved results using the following (abridged) prompt:

```
Classify the similarity between two propositions
as:

(A) HIGHLY RELATED - practically or exactly
the same.
(B) SOMEWHAT RELATED - similar idea or
topic.
(C) DIFFERENT - fundamentally unrelated.

Proposition A: {proposition_a}
Proposition B: {proposition_b}

Respond ONLY with: A, B, or C.
```

Based on the classification outputs, we skip, revise, or add to the GUM.

B.2 Gumbo prompts

B.2.1 Mixed Initiative Interaction. Gumbo generates suggestions to show users, but we can't *show* users all suggestions. To this end, we need to filter what suggestions worth showing. We instantiate mixed-initiative interaction, but this approach requires estimating both the probability of a suggestion being useful and its utility to the user. Here, we use the GUM to generate both the probability of the user selecting the suggestion, and the costs and benefits for the user. We use the following (abridged) prompt:

```
Evaluate each suggestion (1-10 scale):

1. Benefit: How helpful is assistance for
{user_name}?
(1 = not beneficial, 10 = highly beneficial;
consider simplicity, genericness, user's
current actions, urgency.)

2. False Positive Cost: How disruptive
would unsolicited assistance be?
(1 = not disruptive, 10 = highly disruptive;
consider user's workflow and focus.)

3. False Negative Cost: How critical is
assistance if genuinely needed?
(1 = no impact, 10 = significant negative
impact; consider potential setbacks without
help.)

4. Decay: How quickly does the suggestion's
benefit diminish over time?
(1 = immediately obsolete, 10 = remains
useful long-term; consider urgency and task
deadlines.)
```

As context, we provide the suggestion, propositions, and underlying observations in the prompt.

B.2.2 Tool Use. Finally, Gumbo relies on external tools for execution. Here, we use a prompt that selects a subset of tools worth using (with the suggestion / observation in context). An abridged version is below:

```
What tools should you use?
Here are the tools you have at your disposal:

llm (e.g. no tools)
- Generate responses directly without using
any tools.

search
- Search for topics online and provide
citations.
- Use liberally for latest internet information.

filesystem(parameter: (str) filename)
- ONLY use if the file certainly exists on
the user's computer.
- Helpful when viewing the entire file aids
user assistance.

reasoning
- For challenging coding/math problems requiring
deeper thought.

image(parameter: prompt)
- Generate an image given a specific prompt.

Generate a list of useful tools with parameters
in JSON format:
```

C Survey Questions

Below, we outline all the survey questions asked to participants in the email study.

(1) Demographic Information: Name, Age, Gender, Race/Ethnicity, Profession, Email   (Short answer)
(2) How often do you typically check your email?   (Multiple choice: frequency options)
(3) How many different email accounts do you use regularly? (Multiple choice: 1, 2, 3, 4+)
(4) What do you primarily use this specific email for? (Checkboxes: use cases)
(5) Briefly describe the type of content that shows up in this email inbox.   (Paragraph)
(6) How accurately do you feel the propositions shown to you were in reflecting what's in your email? Why do you think they are or aren't accurate?   (Paragraph)
(7) Overall, how accurate were the propositions with high confidence with respect to the context in the email?   (7-point Likert)
(8) Overall, how relevant were the propositions to the context in the email?   (7-point Likert)
(9) Any thoughts about specific propositions? If so, which ones and why?   (Paragraph)
(10) Which data sharing generation was most accurate? (Multiple choice: First, Second, Third)
(11) How much do you agree with the answers to the best data sharing response you picked earlier? Rate these based on accuracy.   (7-point Likert)

(12) Did you feel the propositions themselves (the text) respected your context and privacy? Why or why not? (Paragraph)

D Interview Questions

For our end-to-end evaluation, we conduct an hour-long semi-structured interview. We guide the interview using the questions below. Since the interview is unstructured, we also ask followups.

D.1 Overall Experience

(1) What were your initial expectations for using Gumbo?
(2) How would you describe your overall experience using this?
(3) Was there anything that fell short of your expectations?

D.2 Propositions

(1) How well do you think the propositions understood you?
(2) Can you describe how you felt during looking at the propositions?
(3) Was there anything about the propositions that you particularly liked?
(4) Did you edit any of the propositions?
(5) How accurate did you find the propositions to be?
(6) Were there any propositions that you strongly disagreed with? What was wrong?
(7) Did the propositions reveal something to you that you weren't aware of?

D.3 Suggestions

(1) How well do you think the suggestions understood your goals and concerns based on your interactions?
(2) Did the system offer any insights or recommendations that you found useful?
(3) Did the system offer any insights or recommendations that you did not find useful?
(4) Did you act on its advice?

D.4 General Reflections

(1) If you were able to continue using this system over time, what would you want to use it for?
(2) What features would you add to improve this?
(3) Would you recommend this application to others? What would you tell them about it?
(4) Is there anything else you would like to share about your experience that we haven't already asked?

E Screen Observer Outputs

Our screen observer provides transcriptions that the GUM uses to create propositions. The raw update is entirely in text. The first part of the update is a text-based transcription of the screen (abridged output below), where the first author was editing the Swift frontend for GUMBO:

```
Running Application:
- Xcode (window title: Running Horizon)
```

```
Open Tabs in Xcode:
- Horizon > main
- OnboardingManager
- FrameProcessingService
- AddPropositionPopup
- UserModelPage
- DataModelManagerAPI
- AppDelegate

Open File in Editor:
- File: DataModelManager.swift
- Function: fetchNewProjects(completion:)
[omitted code here]
```

The second part of the transcription is a description of actions the user takes across the past 10 unique frames (anonymized example below):

```
In Figma, the user is designing a UI mockup
titled "User Model" that displays structured
propositions about a user along with confidence
values. They are refining grouped elements
```

and layout, including recommendation cards like "Suit Rentals in Chicago" and "Flights to Chicago".

```
In Overleaf, the user is editing a LaTeX
document for a paper titled "General User
Models." They are working in draft.tex,
using input commands to include section
files, and referencing the Figma UI figure
in "Figure 1." A warning is possibly due
to a formatting issue.
```

```
In Xcode, the user is developing or debugging
a Swift-based application named "Horizon."
They are working in the file DataModelManager
on a function fetchNewProjects(completion:)
that appends suggestions and sends notifs
if certain utility thresholds are met. The
debug console shows a backend communication
error (404 no new updates from the backend)
and a missing symbol warning.
```