A Benchmarking Framework for Context-aware XR Interfaces

Hyunsung Cho
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
hyunsung@cs.cmu.edu

Sarah Yewon Yun
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
ysyun@andrew.cmu.edu

Nancy Ruonan Sun
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
ruonans@andrew.cmu.edu

Ben Lafreniere
Reality Labs Research, Meta
Toronto, Ontario, Canada
ben.lafreniere@gmail.com

Mark Parent
Reality Labs Research, Meta
Toronto, Ontario, Canada
mrkprnt@fb.com

Kashyap Todi
Reality Labs Research, Meta
Redmond, Washington, USA
kashyap.todi@gmail.com

Tanya R. Jonker
Reality Labs Research, Meta
Redmond, Washington, USA
tanyarjonker@gmail.com

Hrvoje Benko
Reality Labs Research, Meta
Redmond, Washington, USA
benko@meta.com

Tongshuang Wu
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
sherryw@cs.cmu.edu

David Lindlbauer
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
davidlindlbauer@cmu.edu

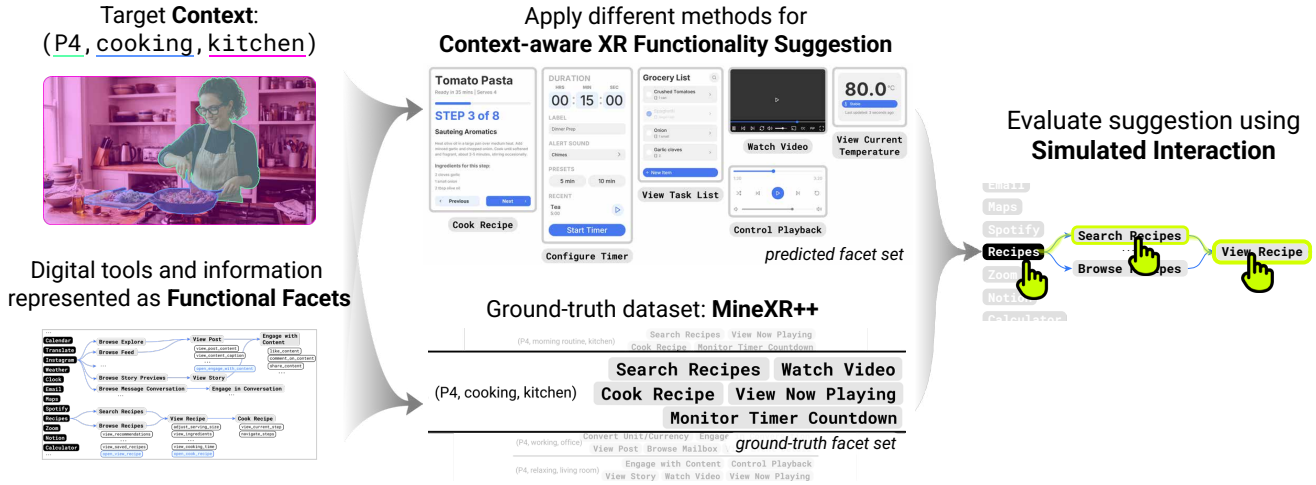


Figure 1: We propose ContextXR, a benchmarking framework for developing and evaluating context-aware functionality suggestion in XR. We contribute (1) a representation of digital tools and information as functional facets, (2) MineXR++, an augmented dataset that serves as ground truth, (3) a problem formulation of the canonical tasks of context-aware suggestion, and (4) an evaluation protocol built on a simulated interaction metric. Jointly, our approach enables principled evaluation of context-aware XR systems.

Abstract

Everyday Extended Reality (XR) systems aim to provide context-aware access to the right functionalities at the right time and place, with minimal manual reconfiguration as users switch context. Yet these interfaces are hard to evaluate: current prototyping and user-study workflows offer no systematic, repeatable way to compare adaptation methods across users and scenarios. We present ContextXR, a novel benchmarking framework for context-aware XR interfaces. ContextXR represents an XR application as a connected graph of *functional facets*, each a semantically coherent group of related capabilities that together support a shared user intent. On this representation, we build MineXR++, a dataset augmenting prior XR interface data with facet-level annotations, and formulate

This work is licensed under a Creative Commons Attribution 4.0 International License.

UIST '26, Detroit, MI, USA

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2856-3/26/11
https://doi.org/10.1145/3830398.3830675

three canonical tasks of context-aware suggestion: context factor analysis, initial facet suggestion, and next facet suggestion. Our evaluation protocol scores suggestion methods by a simulated interaction metric, the navigation and search cost of reaching the desired functionality. Through experiments benchmarking global popularity, relational retrieval, and LLM-based methods, we demonstrate that ContextXR enables the systematic, reproducible evaluation of context-aware XR interfaces.

CCS Concepts

• **Human-centered computing → Mixed / augmented reality**; **Systems and tools for interaction design**; **HCI design and evaluation methods**; *Interactive systems and tools.*

Keywords

Extended Reality, context-aware interfaces, functional suggestion, adaptive user interfaces, computational interaction, benchmarking

ACM Reference Format:

Hyunsung Cho, Sarah Yewon Yun, Nancy Ruonan Sun, Ben Lafreniere, Mark Parent, Kashyap Todi, Tanya R. Jonker, Hrvoje Benko, Tongshuang Wu, and David Lindlbauer. 2026. A Benchmarking Framework for Context-aware XR Interfaces. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST '26), November 02–05, 2026, Detroit, MI, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3830398.3830675

1 Introduction

Extended Reality (XR) is emerging as an everyday companion. Worn continuously through glasses and other head-worn displays, XR can offer situated support, facilitating access to digital content right where a task is happening. While cooking, for instance, the current step of a recipe and a countdown timer can rest on the kitchen counter, in view and within reach as the user works.

A user's context shifts constantly across a day, from cooking to a commute to focused work, and so does the XR functionality that is relevant at each moment. Without a system that adapts automatically, users must rearrange the interface by hand, dismissing functionality from the previous context and then searching through apps to surface what the new context needs. A large body of HCI research has studied how interfaces should adapt to the user, task, and environment to reduce this manual overhead [24, 42, 56]. The goal is for XR to be *context-aware*, using context to surface content relevant to what the user is doing [18].

A core part of context awareness is deciding *what* functionality to surface, and at *what granularity*. Surfacing whole applications exposes more than a situated task requires and quickly clutters the display space [14, 25]. The right unit is instead a working set of *XR functionalities* for the activity at hand—while cooking, the current recipe step, a timer configuration, and a music control, kept available for as long as the task lasts (Figure 1). We frame this as **context-aware functional suggestion in XR**: given a user context, an XR system should infer which functional units ought to be made readily accessible so that desired functionality can be reached with little navigation and search effort.

Despite growing interest in context-aware XR, the field lacks the infrastructure to systematically evaluate and compare solutions. Existing systems are typically bespoke prototypes evaluated through one-off user studies (e. g., [11, 39, 43]). These are essential for demonstrating end-user value, but cannot compare methods across users, scenarios, and modeling strategies in a systematic way, nor isolate which contextual factors actually matter. Other fields have addressed this with shared evaluation frameworks, from computational UI design [16, 62] to computer vision [41, 52] and machine learning [59, 60]. Such frameworks complement user studies, letting developers build and compare solutions against standardized baselines without a costly user study at every iteration. What is missing in context-aware XR is such a benchmarking framework: a shared representation of interface functionality, benchmark tasks reflecting realistic support settings, and evaluation metrics that capture user-facing utility rather than prediction accuracy alone.

In this paper, we present **ContextXR**, a benchmarking framework for studying context-aware functional suggestion in XR. ContextXR represents application functionality as *functional facets*, semantically coherent groupings of atomic capabilities that support a shared user intent. These are coarser than a single capability but finer than a whole application. For example, a music app might expose a `Control Playback` facet, composed of atomic capabilities such as `play`, `pause`, `stop`, or `next song`. A recipe app might expose a `View Recipes` facet, composed of atomic capabilities such as `view_ingredients`, `view_preparation_steps`, and `view_cooking_time`. Functional facets thus capture what an end user is trying to accomplish while remaining grounded in executable interface behavior. Facets within an application are typically connected (e. g., `Browse Recipes` → `View Recipe` → `Cook Recipe`), which we model as an application's *facet graph*. Functional facets and the facet graph serve as the computational units of context-aware XR support for retrieval, suggestion, and evaluation.

We instantiate functional facets in MineXR++, an augmented version of the open-source MineXR dataset [14] annotated with facet-level structure. Over this dataset, we define a set of benchmark tasks for context-aware functionality suggestion in XR: context factor analysis, initial facet suggestion when a user first enters a context, and next facet suggestion during ongoing interaction. Finally, we introduce a simulated interaction protocol centered on *navigation & search cost*, which quantifies the effort a user spends reaching desired functionality given the surfaced suggestions. This protocol moves beyond exact-match retrieval by accounting for partial utility, such as when a suggested facet provides a useful entry point within the correct application.

We demonstrate the framework through benchmark experiments comparing representative approaches on these tasks. These experiments show that ContextXR makes context-aware XR support computationally tractable and systematically comparable, enabling principled study of which contextual factors matter, how different methods compare on the same task, and how much they reduce end-user effort relative to unassisted use.

In summary, we contribute ContextXR, a benchmarking framework for studying context-aware functionality suggestion in XR, combining (1) a new representation of application functionality based on *functional facets*, (2) MineXR++, an augmented dataset with facet-level annotations, (3) benchmark task formulations for context factor analysis, initial facet suggestion and next facet suggestion, and (4) an evaluation protocol centered on navigation &

search cost. Our code, dataset, and benchmark are available at: *augmented-perception.org/publications/2026-contextXR*.

2 Related Work

We situate our work at the intersection of context-aware suggestions in XR, computational approaches for structured UI development, and adaptive XR interfaces. These areas motivate the need for finer-grained representations and the value of structured evaluation for context-aware interface support.

2.1 Context-aware Suggestions in XR

Following Dey and Abowd [18], we treat a system as context-aware if it uses context to provide information or services relevant to the user's task. ContextXR studies one such behavior: using context to suggest relevant XR functionality. Prior work on context-aware suggestions in XR has largely focused on proactive assistance at the level of whole applications, single follow-up actions, or specialized task support. At the application level, COBO [68] studies when suggestions should be presented, Lu et al. [46] examine how reopening a contextually relevant subset of apps supports users across spatial transitions, and Langerak et al. [32] generate tailored explanations for such suggestions. However, application-level access is often too coarse for situated use: users typically need only a task-relevant subset of functionality, placed in context, rather than unrestricted access to entire applications [14, 25].

Other work has explored leveraging contextual information to proactively suggest specific next actions. Systems such as OmniActions [36], ContextAgent [66], Sensible Agent [33], and Satori [34] predict or proactively suggest *single, immediate actions*. For example, when a user is shopping for clothes and viewing a pair of jeans, such systems might suggest actions like *"look up reviews," "search for similar items,"* or *"check price comparisons"*. These approaches solve a different problem. Our setting asks not what the user should do next, but what *set of functionalities* should remain available throughout the activity. In the same scenario, the relevant set might include browsing related products, viewing reviews and product details, comparing prices, messaging a friend for a second opinion, and controlling background music, all surfaced as persistent, spatially arranged entry points. This shifts the problem from immediate action prediction to holistic support, where suggestion quality depends not only on the relevance of individual suggestions but also on the *coverage* the set provides across user needs in context.

This difference in problem formulation also changes evaluation. Prior systems were typically assessed by next-action prediction accuracy or user experience metrics such as task load and usability. These fit immediate assistive actions and interaction techniques, but are less suited to systematically comparing broader functionality suggestions. Our work addresses this gap through a benchmarking framework for context-aware XR functionality suggestion, enabling systematic, reproducible comparison of algorithmic approaches.

2.2 Computational Approaches for Structured UI Development

Outside XR, a substantial body of HCI research treats interface development as a computational problem, supported by large-scale datasets, structured representations, and well-defined tasks for retrieval, evaluation, and adaptation. Datasets such as Webzeitgeist [31], ERICA [17], RICO [16], CLAY [35], and UEyes [27] have enabled data-driven analysis of how interfaces are composed, perceived, and used at scale. Building on such resources, researchers have developed representations that make UI structure computationally accessible, including hierarchical screen parsing [65], graph-based formulations [28], semantic embeddings [37], and text-like encodings of view hierarchies for LLM reasoning [61]. Collectively, these efforts establish interfaces as structured computational objects rather than monolithic visual artifacts.

This structured understanding has enabled downstream work on retrieval and automated evaluation. Embedding-based approaches support retrieval of semantically related screens, components, and commands from screenshots [37, 50], sketches [26], layout structure [5], or natural-language descriptions [1, 62], while other systems use learned structure to score design quality [19, 64] and reconstruct interfaces across devices and form factors [63]. More broadly, this line of work shows the value of explicit representations, reusable datasets, and well-defined computational tasks for making interface development more systematic and comparable. We extend this agenda to XR, where the challenge is not only to represent or adapt interfaces, but to determine which task-relevant functionality to surface across a spatially distributed interface configuration.

2.3 Adaptive XR Interfaces

XR interfaces combine demands that prior work has largely studied separately. Desktop and large-display systems support access to many functions across windows and displays (e. g., [45, 58]), while mobile and ubiquitous computing adapt interfaces to users' changing situations (e. g., [3, 18]). XR extends both challenges: interfaces span many applications and windows distributed throughout the surrounding environment, while their relevance shifts continuously across tasks, activities, and settings [9, 15, 24, 56]. Adaptation is therefore not merely an enhancement, but a necessary means of managing XR's expanded interface space, growing functional complexity, and continuously changing context [42].

Prior work has explored a range of contextual signals for driving such adaptation, including users' cognitive load [43], individual preferences [14, 29, 30], and the semantics and geometry [10, 11, 21, 25] of the environment. More recent systems have also prompted vision-language models to infer relevant situational signals from the user's surroundings [39, 40].

Given these signals, XR interfaces may adapt their *contents* (i. e., what is available and how much), *presentation* (i. e., how and when it is shown), and *placement* (i. e., where it appears) [56]. Much existing work has concentrated on placement and visual configuration: how windows and interface elements should be arranged in space or adjusted to environmental constraints. This focus is well motivated, since XR removes many of the spatial restrictions of conventional UIs and makes layout itself a central design problem. However, this work typically assumes the functionality to present is already given, or selects it only at the granularity of whole applications. Our work complements this line of research by addressing *content*: which functional facets should be surfaced in the first place, before deciding how or where to present them.

3 ContextXR: Benchmarking Framework

ContextXR is a benchmarking framework for evaluating context-aware XR systems across users, tasks, and environments. It enables researchers to answer questions such as which contextual signals are most informative, which modeling strategies perform best, and how much navigation & search cost is reduced through assistance. The framework consists of (1) a functionality-centric representation based on **functional facets**; (2) **MineXR++**, an augmented dataset capturing facet usage across contexts; (3) **problem formulations** for context-aware functional suggestion; and (4) an evaluation protocol centered on **navigation & search cost**.

3.1 Representation: Functional Facets

Our framework uses the functional facet as its central unit of computation.

Why a new representation? Context-aware XR interfaces must decide *what level of functionality* to surface in a limited spatial interface. If the unit is too coarse, such as an entire application, the interface exposes more functionality than the current task and environment require, increasing visual clutter and search effort. If the unit is too fine, such as individual atomic actions or API calls, the interface becomes fragmented and forces users to repeatedly assemble low-level controls to accomplish a single goal.

Applications are naturally composed of many atomic actions. For example, a music app exposes actions like `play`, `pause`, and `shuffle`, and a recipe app exposes actions like `view_ingredients`, `view_preparation_steps`, and `view_cooking_time`. While atomic actions are suitable for direct command-style interaction (e. g., *"Stop the music."* or *"How long does cooking this recipe take?"*), they are often too fine-grained for persistent, visually surfaced XR interfaces. Surfacing capabilities in isolation increases interaction effort, forcing users to reassemble the related capabilities they need.

This motivates an intermediate representation that is more specific than a monolithic application but more usable than individual actions. Our framework addresses this need through *functional facets*. In designing functional facets, we drew on several lines of prior work. Graph-based UI representations, which model interfaces as nodes and transitions, inform the facet graph [28, 65]. Semantic UI representations, which encode the meaning of interface content rather than its structural UI form, inform the intent-level naming of facets [37, 61]. In addition, action-suggestion systems [33, 34, 36, 66] and tool abstractions used by LLM-based agents [49, 53, 67] inform the grounding of capabilities in discrete, invocable units. The specific facet inventory is derived from MineXR [14], whose envisioned XR layouts ground the representation in functionality that users actually wanted surfaced in context.

3.1.1 Functional facets. A *functional facet* is a semantically coherent grouping of one or more related atomic actions, which we call *capabilities*, that together support a shared *user intent*. Each app $a \in \mathcal{A}$ has a set of facets $\mathcal{F}_a$. We represent a facet $f \in \mathcal{F}_a$ as

$$f = (i_f, \mathcal{K}_f),$$

where i_f is the facet's user intent and $\mathcal{K}_f$ is its set of capabilities. Facet names express this intent in a `verb+noun` format. For example, the facet `View Recipe` (Figure 2) serves the intent of viewing a recipe, supported by capabilities such as `view_ingredients`, `view_preparation_steps`, and `view_cooking_time`. Similarly, in a music app, the facet `Control Playback` is supported by capabilities such as `play`, `pause`, `skip_next`, and `shuffle`.

3.1.2 Facet graphs. Facets of the same application are inherently connected through the underlying capabilities. We model these connections using a facet graph

$$G_a = (\mathcal{F}_a, \mathcal{E}_a),$$

where a directed edge $(f_u, f_v) \in \mathcal{E}_a$ exists if facet f_u contains a navigational capability that opens facet f_v. *Navigational capabilities* enable transitions between facets, while other capabilities in $\mathcal{K}_f$ provide information or actions within a facet. Navigational capabilities follow an `open_<facet_name>` convention, reflecting the intent to open the target facet.

Figure 2 illustrates an example. The black nodes labeled with app names represent a special `Open Section` facet, corresponding to each app's entry point (e. g., the app icon) or central navigation hub (e. g., a navigation bar) that links to its main facets. For instance, Instagram connects to `Browse Explore`, `Browse Feed`, `Browse Stories`, and `Browse Message Conversation`, while Recipes connects to `Search Recipes` and `Browse Recipes`. Blue nodes denote navigational capabilities: the edge `open_view_recipe`, for example, connects `Browse Recipes` to `View Recipe`.

The lower portion of Figure 2 grounds this representation in a concrete interface. The `View Recipe` facet is instantiated as a recipe page, where capabilities such as `view_ingredients` and `view_cooking_time` correspond to visible UI elements. The "Start Cooking" button implements `open_cook_recipe`, triggering a transition to the `Cook Recipe` facet, which guides the user through step-by-step instructions via `view_current_step` and `navigate_steps`.

3.2 Dataset: MineXR++

We instantiate this computational framework with MineXR++, an augmented version of the open-source MineXR dataset [14] annotated with the functional facet structure.

MineXR contains 109 personalized XR layouts and 695 widgets from 31 participants across everyday activities such as focus work, chatting with friends, cooking, and relaxing in four environments: office, living room, kitchen, and coffee shop. Participants created these layouts by taking screenshots from apps and websites, or drawing sketches, and cropping the regions they wanted to place in the environment as custom widgets.

While MineXR offers a valuable view into envisioned everyday XR use, its original form is not directly suitable for computational evaluation. Widgets are free-form image crops, and the accompanying annotations do not provide a consistent structure for representing functionality across apps and scenarios.

To operationalize the dataset within our framework, we augmented MineXR with a functional-facet annotation layer. We first constructed a functional facet database. Facet definitions were derived from MineXR screenshots and additional screenshots from recent app versions, and organized around recurring user intents and their associated capabilities. To reduce sparsity, applications

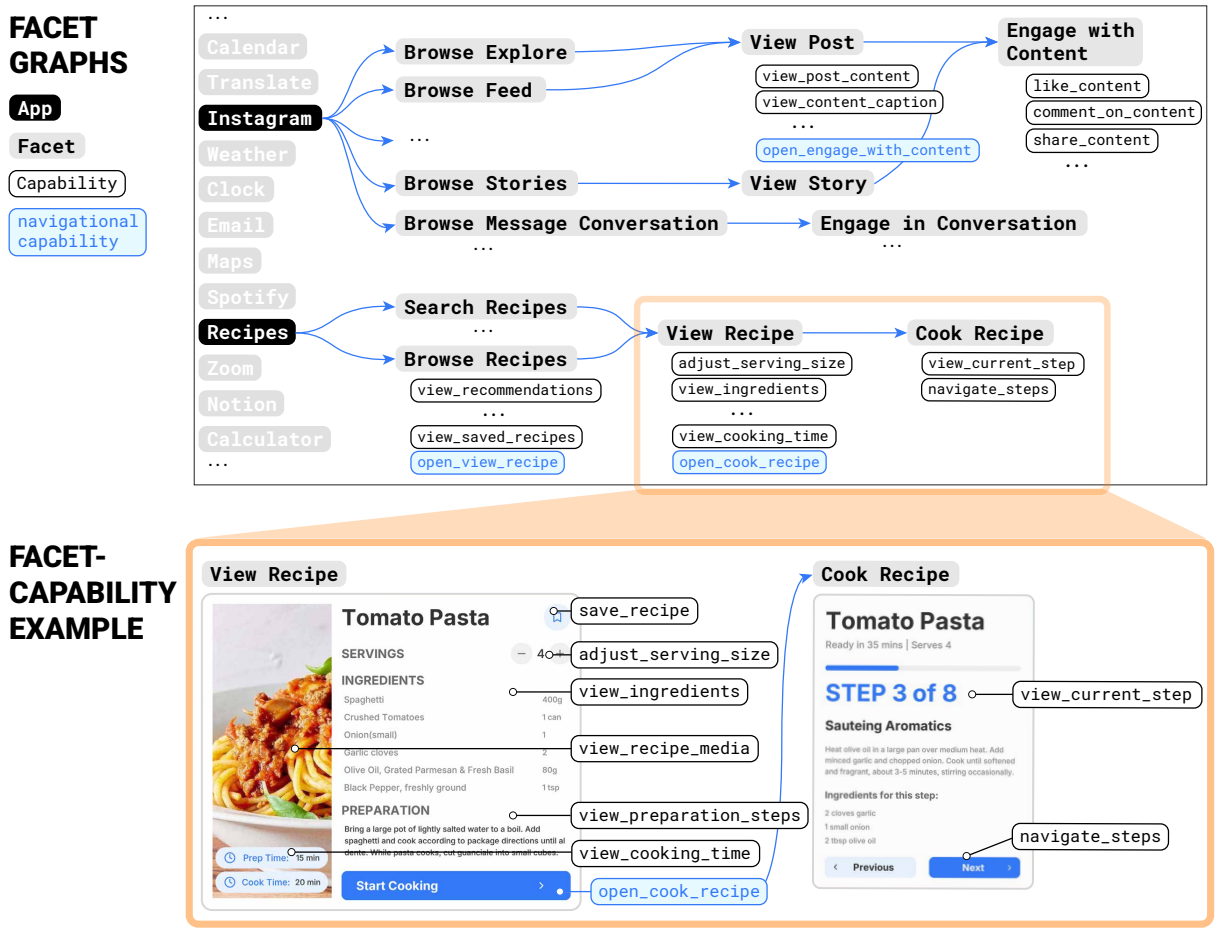


Figure 2: Top: Snippet of app–facet–capability structure. Bottom: Visualized example of facets and their component capabilities. A navigational capability (e. g., `open_cook_recipe`) leads to another facet (e. g., `Cook Recipe`).

with highly similar functionality and limited usage were consolidated into shared categories. The resulting database covers 42 applications, comprising 273 facets and 1,007 capabilities, of which 101 facets were placed by participants during the MineXR data collection.

Using this database, we annotated each widget with its corresponding functional facet(s) and the specific capabilities it exposed in context. Annotation criteria were iteratively discussed among three authors; primary annotation was conducted by one author and reviewed by a second author. We refer to this facet-annotated version as MineXR++.

In summary, each facet contains information about what app it originates in and which capabilities it is composed of. Each placed widget in MineXR++ links a facet to its context of use: user ID (e. g., "P07"), task (e. g., "cooking"), and location (e. g., "kitchen"), as shown in the schema in Figure 3, which also serves as the structure our relational retrieval method traverses (Section 4). These fields are the minimum requirements for a dataset to be compatible with our proposed benchmarking framework, which in the future could be augmented with information about placement, grouping, user state (e. g., cognitive load) or user capabilities.

3.3 Problem Formulation

We formulate context-aware functionality suggestion in XR as the problem of deciding *which functional facets to make readily accessible* for a given usage context. The goal is not to predict a single next action, but to identify the subset of functionality that should be surfaced so that users can reach what they need with low friction.

Our benchmark operationalizes context as a triple of user identity u, task t, and environment e, following the structure of MineXR [14]. This lets us study how these factors relate to the functionality a user needs. For a given context $q = (u, t, e)$, the desired outcome is a target set of functional facets, $Y_q \subseteq \mathcal{F}$, where $\mathcal{F}$ is the set of candidate facets across applications. Intuitively, Y_q is the functionality that should be easy to reach in that situation. In MineXR++, Y_q is the set of facets annotated on the XR elements that participants placed in their envisioned XR layouts. A prediction method produces a subset $\hat{Y}_q \subseteq \mathcal{F}$, the facets it suggests for that context, which we evaluate against the ground-truth target Y_q.

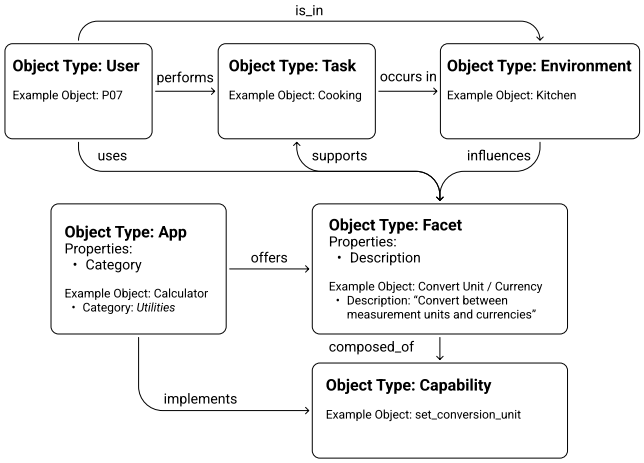


Figure 3: Object schema relating functional facets to their app, capabilities, and context of use (user, task, environment). This schema defines the structure of MineXR++ and is traversed by the relational retrieval method in Section 4.

This formulation is agnostic to how $\hat{Y}_q$ is produced: a system may retrieve it from similar examples, infer it from language or structured context, generate it with an LLM, or combine multiple strategies. What matters for the framework is that different methods can be evaluated on the same underlying task, surfacing the right functional facets for the right context.

3.4 Evaluation Protocol: Simulated Interaction

To evaluate suggestions, ContextXR models the **navigation & search cost** required to access the desired facet set Y_q given a suggested set $\hat{Y}_q$.

Why navigation & search cost? To evaluate context-aware functionality suggestion in XR, we aim to measure how much effort suggestions save a user in reaching the desired set of functional facets, rather than relying solely on set overlap with ground-truth facets. In XR, usefulness depends not only on exact correctness but on how suggestions reduce navigation and search effort. For example, a closely related facet from the correct application may still be substantially more useful than an unrelated one, even if both would be counted as incorrect under exact-match metrics. Crucially, this requires evaluating *sets* of suggested functionality: systems surface multiple facets at once, and users can choose among them, so utility depends on how the *entire set* supports access. Whereas top-k metrics such as Recall@K or Hit@K reduce predictions to binary correctness at a cutoff, navigation & search cost captures partial relevance and how a set of suggestions jointly reduces user effort, and enables direct comparison against an unassisted manual baseline. In taking access effort rather than prediction accuracy as the unit of evaluation, we follow menu-optimization work that scores interface designs by the time or steps a user spends reaching a target [2, 54]. We adapt this idea to functional suggestion, using navigation & search cost as a complement to the retrieval and prediction metrics above rather than a replacement for them.

3.4.1 Navigation & search cost. The navigation & search cost model measures the effort a user spends reaching the desired set of functional facets, following a simple simulated user strategy. It is a deliberately simplified model of access: a user opens a menu, scans for the relevant application or suggestion, and navigates to the target facet, with each step assigned a fixed cost. This abstracts away much of the perceptual, motor, and cognitive variation of real XR interaction, and is intended for comparing methods under identical assumptions rather than estimating absolute human performance. The framework is designed so that this layer can be replaced with richer cost models, discussed in Section 5.3.

Manual baseline. Without system support, the user opens the application menu and navigates to each desired facet without any system support. The total manual cost is

$$C_{\text{manual}}(Y_q) = c_{\text{open_menu}} + \sum_{f_i \in Y_q} \Big(c_{\text{app_scan}}(f_i) + c_{\text{facet_nav}}(f_i) \Big),$$

where $c_{\text{open_menu}}$ is the one-time cost of opening the app list, $c_{\text{app_scan}}(f_i)$ is the cost of locating the relevant application for facet f_i, and $c_{\text{facet_nav}}(f_i)$ is the cost of navigating within that application G_a to reach f_i.

Suggestion-assisted access. When a system suggests $\hat{Y}_q$, each target facet in Y_q is resolved into one of three disjoint subsets:

- H: exact hits, where the desired facet itself is directly surfaced
- A: same-app adjustments, where a surfaced facet from the same application provides a useful entry point but is not itself the desired facet
- M: misses, where no helpful suggestion is available and the user must fall back to manual access.

These subsets partition the target set: $Y_q = H \cup A \cup M$.

Exact hits are directly accessible and incur no additional navigation cost. For same-app adjustments, we assume that the surfaced facet places the user within the correct application, so the remaining effort is smaller than full manual access but still non-zero. This residual effort is captured by an adjustment cost. Misses require falling back to manual navigation, and the menu is opened only when at least one miss occurs.

The suggestion-assisted cost is

$$C_{\text{suggest}}(Y_q, \hat{Y}_q) = c_{\text{suggest_scan}}(\hat{Y}_q) + \sum_{f_i \in A} c_{\text{adjust}}(f_i) + \sum_{f_i \in M} \Big(c_{\text{app_scan}}(f_i) + c_{\text{facet_nav}}(f_i) \Big) + \mathbf{1}_{|M|>0}\, c_{\text{open_menu}},$$

where $c_{\text{suggest_scan}}(\hat{Y}_q)$ is the cost of scanning the surfaced suggestions, $c_{\text{adjust}}(f_i)$ is the residual cost of reaching f_i from a same-app suggestion, and the final terms capture fallback to manual access for misses. A lower navigation & search cost indicates that the surfaced subset provides more efficient access to the desired functionality.

3.4.2 Supplementary retrieval metrics. To support comparison with standard retrieval and recommendation settings, the framework can also report traditional metrics such as Recall@K, F1@K, and Hit@K. We treat these as complementary measures of retrieval-style correctness, while navigation & search cost captures user-facing access efficiency.

4 Experiments

We apply ContextXR in three experiments, each targeting a canonical question that arises when developing a context-aware XR system: (1) a **context factor analysis** asking *which contextual signals matter*—how much a user's identity, task, and environment each explains the functionality they need; (2) an **initial facet suggestion** task asking *what to surface when a user first enters a context*, before any interaction has unfolded; and (3) a **next facet suggestion** task asking *what to surface next as the interaction progresses*, using the functionality already in use as additional evidence. These illustrate how ContextXR can support systematic, reproducible evaluation, spanning the understanding of context, cold entry into a context, and ongoing interaction within it.

4.1 Experiment 1: Context Factor Analysis

4.1.1 Task. Before deciding what to surface, a developer needs to know which context factors actually shape a user's needs: does desired functionality track *who* the user is, *what* they are doing, *where* they are, or some combination? This experiment quantifies, for each facet, which of the three context factors (user, task, environment) most strongly predicts its use, using the human-authored target sets in MineXR++ as ground truth.

4.1.2 Methods and procedure. To examine how desired functionality depends on context, we performed a *lift*-based analysis [57] at the facet level. For a given context factor, lift compares how often a facet appears under a matched condition (for example, the same user, task, or environment) relative to how often it appears overall in the dataset. A lift greater than 1 indicates that the facet is more likely than expected under that condition. For each facet, we identified which context factor produced the strongest lift, i. e., the factor most associated with its use.

4.1.3 Results. The left of Figure 4 shows the distribution of dominant factors across facets. Among facets used in MineXR++, 47 were user-dominant, 14 task-dominant, and 12 environment-dominant, while 8 showed mixed dependence and 20 showed no lift above baseline. This variability suggests that desired functionality in XR does not reduce to a single factor alone, but reflects a combination of personal preference, task relevance, and environment setting.

The example lift profiles on the right side of Figure 4 show one representative facet for each pattern. `Duolingo: Complete Lesson` is user-dominant, `Books: Browse Library` is task-dominant, and `Netflix: Browse Content` is environment-dominant. `Reminders: Browse Task Lists` shows a mixed profile, with similarly strong user and task lift. These examples indicate that different facets align with different contextual axes, motivating suggestion methods that can incorporate different kinds of prior evidence. The facet representation makes this analysis possible, offering a way to examine which factors drive functional needs in context-aware XR.

4.2 Experiment 2: Initial Facet Suggestion

4.2.1 Task. When a user first enters a context q, an XR system must decide what functionality to surface before any interaction unfolds. We frame this as *initial facet suggestion*: given the current user, task, and environment, predict a set of facets $\hat{Y}_q$ to make readily accessible at the start of the episode. The target Y_q is the

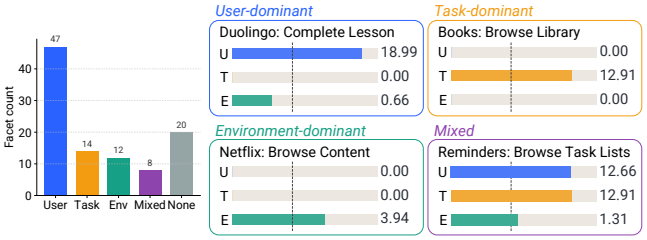


Figure 4: Left: distribution of the dominant context factor across facets (user-, task-, environment-dominant, mixed, or no lift). Right: one representative facet per pattern, showing lift by context factor.

set of facets the participant placed for that context in MineXR++, so a method is scored on how well it recovers a real user's situated selection. This setting captures cold-start and example-augmented support, and evaluates how far different methods reduce the effort required to reach desired functionality relative to manual access.

4.2.2 Methods. We use ContextXR to compare five strategies: manual access, global popularity, relational retrieval, zero-shot LLM, and few-shot LLM. These span context-agnostic baselines, structured retrieval, and generative reasoning, chosen to represent the method families used in adaptive XR rather than to reimplement any single prior system. We include the LLM conditions to represent the recent class of context-aware assistance systems that use large language models to reason over sensed context [33, 36, 66]. Because those systems predict single next actions rather than sets of persistent functionality (Section 2), we do not compare against them directly; instead, our LLM baselines apply the same underlying contextual-reasoning capability to our set-based task, so that generative and non-generative approaches can be compared under one benchmark. For all assisted methods, the system returns a top-10 set of suggested facets, evaluated with both navigation & search cost and standard retrieval metrics.

Manual access. Manual access is the unassisted reference condition: no facets are proactively suggested, and the user reaches desired functionality through manual navigation. This is computed using the simulated interaction procedure of Section 3.4.1.

Global popularity. As a simple low-cost baseline, global popularity ignores the query context and ranks facets only by how often they appear in the training data. This provides a context-agnostic prior over commonly needed functionality.

Relational retrieval. Relational retrieval uses a structured context–facet representation (Figure 3). Each dataset entry is a context–facet event, from which we construct a typed entity–relation schema with nodes for users, tasks, environments, apps, facets, and capabilities, linked by fixed relations such as user-*performs*-task, user-*uses*-facet, and facet-*composed-of*-capability. The schema provides structured retrieval by defining the entities and relations over which counts, adjacency maps, and capability links are computed. Given a query context $q = (u, t, e)$, the model ranks candidate facets using context-conditioned statistics from the training data, combining signals such as $P(f|u)$, $P(f|t)$, $P(f|e)$,

Table 1: Average navigation & search cost and retrieval quality for *initial facet suggestion*. Parentheses indicate the number of evaluated scenarios. Cost reductions are relative to manual access computed on the same scenarios.

Condition	Cost	Recall@10	F1@10	Hit@10
Manual Access	21.20	–	–	–
Global Popularity	19.85 (-6.4%)	0.167	0.132	0.688
Relational Retrieval	18.39 (-13.3%)	0.259	0.193	0.798
Zero-shot LLM	17.66 (-16.7%)	0.228	0.168	0.761
Few-shot LLM				
User *(106)*	16.33 (-22.6%)	**0.338**	**0.255**	0.830
Task *(40)*	18.41 (-15.9%)	0.274	0.217	0.825
Environment *(108)*	17.07 (-18.8%)	0.287	0.206	**0.870**
User+Task *(2)*	**15.25 (-30.7%)**	**0.464**	**0.399**	**1.000**
User+Env *(94)*	**15.53 (-22.7%)**	0.319	0.229	0.862
Task+Env *(47)*	19.54 (-12.6%)	0.222	0.178	0.702

Table 2: Average navigation & search cost and retrieval quality for *next facet suggestion*. Scenarios are weighted equally after averaging over their timelines. Cost reductions are relative to manual access computed on the same scenarios.

Condition	Cost	Recall@10	F1@10	Hit@10
Manual Access	21.19	–	–	–
Global Popularity	18.70 (-11.7%)	0.168	0.089	0.466
Relational Retrieval	15.70 (-25.9%)	**0.355**	**0.162**	**0.687**
Zero-shot LLM	16.00 (-24.5%)	0.205	0.102	0.488
Few-shot LLM				
User	**14.61 (-31.1%)**	0.285	0.147	0.574
Task	19.22 (-9.3%)	0.079	0.044	0.188
Environment	15.82 (-25.3%)	0.209	0.104	0.492
User+Task	20.98 (-1.0%)	0.007	0.004	0.014
User+Env	15.83 (-25.3%)	0.257	0.122	0.494
Task+Env	19.30 (-8.9%)	0.090	0.046	0.205

$P(f|u,t)$, $P(f|u,e)$, and $P(f|t,e)$. To remain robust when exact matches are sparse, it also applies soft transfer over task and environment labels using cosine similarity between Sentence-BERT embeddings from `all-MiniLM-L6-v2` [51].

This method is motivated in part by XR deployment constraints. Compared with LLM-based inference, retrieval over precomputed relational statistics is substantially cheaper and lower-latency, offering a favorable accuracy–cost tradeoff for resource-constrained headsets, where compute and battery use directly affect usability.

LLM-based suggestion. The two LLM conditions operate over a structured prompt (Appendix A) containing the current scenario (task and environment) and a catalog of candidate facets with descriptions, capabilities, and facet-graph links.

The **zero-shot LLM** predicts relevant facets from the current context and catalog alone, without retrieved examples. The **few-shot LLM** extends this through in-context learning [4], adding historical scenarios and their facet selections under controlled transfer conditions (e. g., matching user, task, or environment).

In both cases, the model selects a fixed-size set of facets (top-10), balancing direct task support, complementary functionality, and ambient utilities, and using the facet graph to reason about granularity and avoid redundant navigation chains. All LLM results use Gemini 3 Flash at temperature 0.2.

4.2.3 Procedure. Each method is evaluated on every eligible context q in MineXR++, producing a top-10 suggested set $\hat{Y}_q$ that we compare against the human-authored target Y_q. Global popularity and relational retrieval use seeded five-fold cross-validation at the scenario level, so that all observations from a held-out scenario are excluded from its training fold. For the few-shot conditions, we report results separately for each transfer condition (e. g., *User*, *Task*, *User+Env*), drawing historical examples only from scenarios matching the specified factor(s); parentheses in Table 1 give the number of evaluated scenarios for which such matched examples exist. We report navigation & search cost as the primary measure, expressed as the percentage reduction relative to manual access on the same scenarios, alongside standard top-10 retrieval metrics (Recall@10, F1@10, Hit@10) as complementary measures of retrieval-style correctness.

4.2.4 Results. Table 1 summarizes the results. All assisted methods reduce navigation & search cost relative to manual access, showing that even simple proactive support can make desired functionality easier to reach, though the benefit varies by suggestion strategy and by what prior information is available.

Global popularity yields only a modest improvement (−6.4%), suggesting that frequently used facets alone do not characterize what should be surfaced in a specific context. Relational retrieval improves more substantially (−13.3%), indicating that structured, context-conditioned historical signals capture more of the context-to-functionality mapping than frequency alone. Zero-shot LLM performs better still (−16.7%), implying that general contextual reasoning already captures part of the mapping without task-specific statistics.

The strongest results come from few-shot LLM conditions, though their effectiveness depends strongly on which context factors are matched. *User+Task* is best numerically, but with only two matched scenarios we interpret it cautiously. Among more broadly supported settings, *User* and *User+Env* perform best, reducing navigation & search cost by −22.6% and −22.7% respectively. This suggests that user-aligned examples are especially valuable for initial facet suggestion, a pattern partially consistent with the lift analysis (Section 4.1), where many facets showed strong user-dependent lift. By contrast, *Task+Env* underperforms the other pairwise settings, further suggesting that user-specific regularities matter most for initial facet suggestion in our data.

Qualitative observations. Figure 5 illustrates a concrete example. For the context ("P04", "relaxing", "living room"), the ground truth includes nine facets from Spotify, Uber Eats, Instagram, and Amazon. Suggestions for other apps are omitted from this figure, and `Open Section` denotes an app's entry point or central navigation hub.

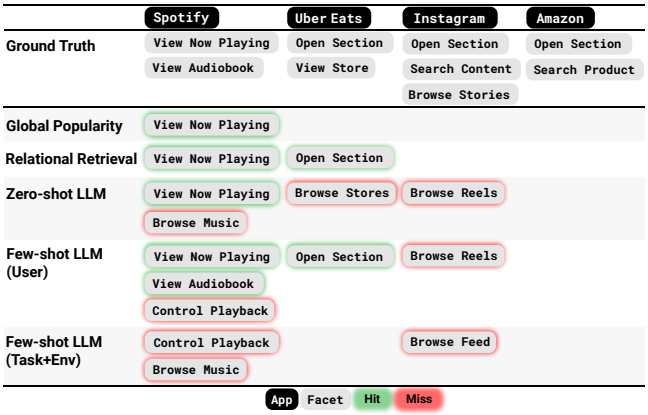


Figure 5: Example of initial facet suggestion for a specific participant and context ("P04", "relaxing", "living room"). By comparing ground-truth facets in MineXR++ with suggestions from multiple methods, ContextXR makes differences in functional coverage explicit for analysis.

Global popularity surfaces only `Spotify: View Now Playing`, missing the relevant `Uber Eats`, `Instagram`, and `Amazon` facets. Relational retrieval adds `Uber Eats: Open Section` but still misses `Instagram` and `Amazon`. Zero-shot LLM improves app-level grounding by suggesting the right apps, but often at the wrong facet granularity, e. g., `Spotify: Browse Music` instead of `View Audiobook` and `Uber Eats: Browse Stores` instead of `Open Section`. *User* few-shot prompting better captures this user's habitual choices, recovering both `Spotify: View Audiobook` and `View Now Playing` as well as `Uber Eats: Open Section`. *Task+Env* examples are less helpful here, reflecting other users' preferences, such as `Netflix` and `Webtoon`, rather than this person's typical selections.

These observations reinforce the lesson of Experiment 1: desired functionality in XR is shaped by multiple context factors, and methods that selectively incorporate the right prior evidence perform best. They also show how ContextXR makes it possible to compare not only *which model* performs better, but also *which kinds of context information* are most useful for reducing navigation & search cost at the start of an interaction episode. Using ContextXR, new methods can be compared with existing ones in a systematic, reproducible way.

4.3 Experiment 3: Next Facet Suggestion

4.3.1 Task. As interaction unfolds, systems can leverage the facets already used in the current episode to infer what functionality is likely to be needed next. In the *next facet suggestion* task, the model predicts the next desired facet at each step of a scenario timeline, capturing step-by-step adaptive support rather than one-shot initialization. Here the facets a user has already placed serve as additional contextual evidence: the task tests whether within-episode interaction history improves prediction beyond the static context $q = (u, t, e)$ alone.

4.3.2 Methods. We evaluate the same family of methods as in Experiment 2, adapted to predict at each interaction step rather than only at scenario entry. Manual access again serves as the unassisted reference. Global popularity remains context-agnostic. Relational retrieval uses the same structured representation, now conditioned on the currently observed context together with the active facet priors at that step. The zero-shot and few-shot LLM conditions receive the current context and the interaction history and suggest the next facet to present. For few-shot prompting, we again vary which prior examples are matched to the current step by user, task, environment, or their pairwise combinations. All methods make 10 suggestions per step.

4.3.3 Procedure. Following sequence-style evaluation, we compute per-step predictions along each scenario timeline and average at the scenario level so that long scenarios do not dominate the aggregate. We report both navigation & search cost and standard top-10 retrieval metrics, using the same simulated interaction procedure.

4.3.4 Results. Table 2 reports the results for next facet suggestion. As in Experiment 2, the assisted methods reduce navigation & search cost relative to manual access, but the pattern across baselines differs in informative ways.

Global popularity again provides only a modest improvement (−11.7%), suggesting that a generic frequency prior captures only part of the structure needed for sequential support. Relational retrieval improves more clearly (−25.9%), achieving the highest Recall@10, F1@10, and Hit@10. This indicates that structured inter-facet relations become particularly useful once the task is interactive and a richer prior is available. Zero-shot LLM performs comparably (−24.5%), showing that sequence-aware contextual reasoning can capture some progression structure even without retrieved examples.

The best overall performance comes from few-shot LLM with *User*-matched examples, at the lowest navigation & search cost (−31.1%). This suggests that next-step functional needs are particularly personalized: once interaction is underway, knowing how a given user tends to proceed appears more informative than the nominal task or environment. *Environment*-only and *User+Env* examples also perform well, each reducing navigation & search cost by about 25%, again pointing to situational regularities beyond task labels.

By contrast, *Task*-based transfer is much less effective in this sequential setting. *Task*-only and *Task+Env* produce only small gains, and *User+Task* performs poorly, with almost no gain over manual access. This likely reflects data sparsity and over-specific matching in the sequential regime: when few closely matched trajectories exist, the examples may transfer poorly and even mislead the model.

Qualitative observations. The same context from Experiment 2 illustrates this difference. For ("P04", "relaxing", "living room"; see Figure 5 for ground truth). Sequential methods, such as relational retrieval, zero-shot LLM, and few-shot LLM with *User* and/or *Env* matches, surface facets missed at initial facet suggestion, including `Amazon: View Product Details`, `Search Product`, and `Open Section`. *Task*-based few-shot conditions remain influenced by other users' preferences from similar contexts, and repeated next-step prediction amplifies these mismatches over the scenario.

These results complement Experiment 2 by showing that the most useful source of signal changes once interaction is underway. Initial facet suggestion benefits most from matched prior examples, especially user-aligned ones, whereas next facet suggestion can draw much more heavily on evidence revealed by the current trajectory. More broadly, they show how ContextXR supports not only systematic comparison of methods across tasks, but also analysis of how different sources of evidence—global priors, structured history, matched examples, and within-episode interaction—contribute under different adaptive-support settings.

5 Discussion

In the following, we discuss the implications of our experiments for building context-aware XR systems, and how ContextXR can be used and extended.

5.1 Takeaways and Design Recommendations

Our findings suggest several concrete directions for researchers and practitioners designing context-aware XR interfaces.

Prioritize user modeling. Across facets, functional needs were user-dominant far more often than task- or environment-dominant (Experiment 1), and user-matched examples gave the strongest gains for initial suggestion (Experiment 2). For deciding *what* to surface, personalization is therefore among the highest-leverage signals, and systems that model individual users are likely to outperform those that adapt only to task or location.

Shift the signal source as interaction unfolds. The most useful evidence changes over a session: prior user history helps most at cold start, but within-episode interaction becomes the dominant signal once the user begins acting (Experiments 2 and 3). Rather than applying a static context model throughout, a deployed system should lean on user-specific priors at entry and shift toward the live interaction trajectory as it accumulates.

Combine context factors rather than relying on one. No single factor accounted for functional needs on its own; user, task, and environment each dominated for different facets (Experiment 1). Systems should fuse these signals rather than key off any one, and the relative weight of each is itself something ContextXR can help calibrate on a target dataset.

Enrich functional structure at the source. Our facets are annotated from existing applications, but richer and more reliable functional structure could come from several routes: app developers declaring functional facets or tags at development time; automated semantic parsing and recognition at the platform or infrastructure level, for those with such access; or generative-interface systems that expose functional units as they synthesize UI [6–8]. Investing in this structure, by any of these routes, would make applications more amenable to intelligent, context-aware composition in XR.

5.2 Using the Framework

ContextXR is released as an open-source benchmark with example Python scripts for the three tasks presented in this paper. It takes context as input, defined in our benchmark as a triple of user, task, and environment, represented as strings, and evaluates how well a method maps that context to the functionality a user needs. Context sensing is outside the framework's scope. Practitioners can use their own sensing modules—for example, an activity recognition model or a vision-language model—as long as they produce user, task, and environment labels that then feed into MineXR++ and the methods. Given this context as input, the framework outputs navigation & search cost and retrieval metrics for the new method, enabling comparison against the benchmark methods.

5.3 Extending the Framework

ContextXR is designed to be extended along several fronts.

Context types. The context representation can grow beyond the user–task–environment triple to include factors such as mobility, social setting, or device type, each of which plausibly shapes what functionality should be surfaced. Because methods consume context abstractly, adding a context type requires no change to the evaluation itself.

Data. MineXR is derived from lab-based, envisioned layouts rather than long-term in-the-wild use, which limits scale and leaves user- and context-specific signals sparse. With larger and more diverse data, ContextXR can accommodate techniques that exploit cross-user regularities, such as collaborative filtering and hybrid recommenders, as well as richer ontological representations of users, tasks, and environments. Synthetic data from generative agents or context simulators offers a complementary path to expand coverage and stress-test methods on rare or counterfactual scenarios.

Tasks. New benchmark tasks, for example longer-horizon prediction or multi-user settings, can be defined over the same facet representation and scored with the same protocol, so that results remain comparable across studies.

Metrics. Our navigation & search cost model captures navigation and search effort under a deliberately simplified simulation. This layer can be replaced with richer cost models: biomechanically grounded costs from movement simulation [20, 23, 47, 48], attention- and cognition-aware costs for the effort of noticing and switching among elements [12, 13, 38, 44], and costs conditioned on a user's familiarity and a device's input and display characteristics [22, 55]. Pairing these with qualitative or experience-centered evaluation, including approaches that model user perception and affect [69], would give a fuller account of interface quality than efficiency alone.

Methods. Beyond the baselines studied here, ContextXR invites richer suggestion methods, for example recommender-style models that exploit cross-user structure or LLM agents that reason over the facet graph, each directly comparable to our results.

5.4 Limitations

ContextXR emphasizes quantitative, reproducible evaluation, which necessarily abstracts away aspects of experience such as perceived relevance, cognitive load, trust, and interface coherence. The navigation & search cost model is a simplified simulation with fixed per-step costs and a uniform user strategy, and does not capture the full perceptual, cognitive, and physical dynamics of real XR interaction; richer cost models are a natural extension, as discussed

above. Finally, our evaluation assumes context is given rather than sensed, so it does not account for upstream recognition error; evaluating methods under imperfect, sensed context is an important direction for assessing real-world robustness.

6 Conclusion

We presented ContextXR, a benchmarking framework for studying context-aware functional suggestion in XR. By introducing functional facets as an intermediate representation of application functionality, augmenting MineXR with facet-level annotations to produce MineXR++, and defining benchmark tasks for context-factor analysis and initial and sequential facet suggestion, the framework makes context-aware XR interface support systematically comparable across methods, data, and metrics. Across three experiments, we showed how ContextXR supports not only comparison of methods, but also analysis of what drives effective support. We found that desired functionality depends on multiple context factors, and that the usefulness of different signals—global priors, matched prior examples, and within-episode interaction history—varies across adaptive-support settings. By grounding evaluation in navigation & search cost, the framework connects model performance to user-facing effort, moving beyond exact-match metrics toward a more practical notion of utility. More broadly, ContextXR offers a foundation for cumulative progress in context-aware XR systems. By providing a shared representation, tasks, and evaluation, it lets researchers and developers study how different forms of context, data, and modeling approaches help reduce the effort of functional access in XR.

Acknowledgments

The work in this paper was supported in part by the National Science Foundation under Award No. IIS-2542186. We thank Jeongin Park for her help with the figures.

References

[1] Eytan Adar, Mira Dontcheva, and Gierad Laput. 2014. CommandSpace: modeling the relationships between tasks, descriptions and features. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 167–176.

[2] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. Menuoptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 331–342.

[3] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. 2007. A survey on context-aware systems. *International Journal of ad Hoc and ubiquitous Computing* 2, 4 (2007), 263–277.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual search for mobile user interface design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[6] Yining Cao, Peiling Jiang, and Haijun Xia. 2025. Generative and malleable user interfaces with generative and evolving task-driven data model. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–20.

[7] Jiaqi Chen, Yanzhe Zhang, Yutong Zhang, Yijia Shao, and Diyi Yang. 2026. Generative interfaces for language models. In *Findings of the Association for Computational Linguistics: ACL 2026*. 1499–1519.

[8] Ruijia Cheng, Titus Barik, Alan Leung, Fred Hohman, and Jeffrey Nichols. 2024. BISCUIT: Scaffolding LLM-Generated Code with Ephemeral UIs in Computational Notebooks. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 13–23.

[9] Yi Fei Cheng, Ari Carden, Hyunsung Cho, Catarina G. Fidalgo, Jonathan Wieland, and David Lindlbauer. 2025. Augmented Reality Productivity In-the-Wild: A Diary Study of Usage Patterns and Experiences of Working With AR Laptops in Real-World Settings. *IEEE Transactions on Visualization and Computer Graphics* 31, 10 (2025), 9195–9212. doi:10.1109/TVCG.2025.3592962

[10] Yi Fei Cheng, Christoph Gebhardt, and Christian Holz. 2023. InteractionAdapt: Interaction-driven workspace adaptation for situated virtual reality environments. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[11] Yi Fei Cheng, Yukang Yan, Xin Yi, Yuanchun Shi, and David Lindlbauer. 2021. SemanticAdapt: Optimization-based adaptation of mixed reality layouts leveraging virtual-physical semantic connections. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 282–297.

[12] Hyunsung Cho, Xuejing Luo, Byungjoo Lee, David Lindlbauer, and Antti Oulasvirta. 2026. Simulating Human Audiovisual Search Behavior. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems* (Barcelona, Spain) *(CHI '26)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3772318.3790614

[13] Hyunsung Cho, Alexander Wang, Divya Kartik, Emily Liying Xie, Yukang Yan, and David Lindlbauer. 2024. Auptimize: Optimal Placement of Spatial Audio Cues for Extended Reality. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 126, 14 pages. doi:10.1145/3654777.3676424

[14] Hyunsung Cho, Yukang Yan, Kashyap Todi, Mark Parent, Missie Smith, Tanya R Jonker, Hrvoje Benko, and David Lindlbauer. 2024. MineXR: mining personalized extended reality interfaces. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.

[15] Shakiba Davari and Doug A. Bowman. 2024. Towards Context-Aware Adaptation in Extended Reality: A Design Space for XR Interfaces and an Adaptive Placement Strategy. arXiv:2411.02607 [cs.HC] https://arxiv.org/abs/2411.02607

[16] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 845–854.

[17] Biplab Deka, Zifeng Huang, and Ranjitha Kumar. 2016. ERICA: Interaction mining mobile apps. In *Proceedings of the 29th annual symposium on user interface software and technology*. 767–776.

[18] Anind K Dey and Gregory D Abowd. 1999. Towards a better understanding of context and context-awareness. https://www.cs.umd.edu/class/spring2018/cmsc818G/files/contextdey.pdf

[19] Peitong Duan, Chin-Yi Cheng, Gang Li, Bjoern Hartmann, and Yang Li. 2024. UICrit: Enhancing Automated Design Evaluation with a UI Critique Dataset. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. ACM, Pittsburgh PA USA, 1–17. doi:10.1145/3654777.3676381

[20] João Marcelo Evangelista Belo, Anna Maria Feit, Tiare Feuchtner, and Kaj Grønbæk. 2021. XRgonomics: Facilitating the Creation of Ergonomic 3D Interfaces. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 290, 11 pages. doi:10.1145/3411764.3445349

[21] Andreas Fender, David Lindlbauer, Philipp Herholz, Marc Alexa, and Jörg Müller. 2017. Heatspace: Automatic placement of displays by empirical analysis of user behavior. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 611–621.

[22] Krzysztof Gajos and Daniel S Weld. 2004. SUPPLE: Automatically Generating User Interfaces. (2004), 93–100.

[23] Eric J Gonzalez and Sean Follmer. 2023. Sensorimotor simulation of redirected reaching using stochastic optimal feedback control. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–17.

[24] Jens Grubert, Tobias Langlotz, Stefanie Zollmann, and Holger Regenbrecht. 2017. Towards Pervasive Augmented Reality: Context-Awareness in Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* 23, 6 (2017), 1706–1724. doi:10.1109/TVCG.2016.2543720

[25] Violet Yinuo Han, Hyunsung Cho, Kiyosu Maeda, Alexandra Ion, and David Lindlbauer. 2023. BlendMR: A computational method to create ambient mixed reality interfaces. *Proceedings of the ACM on Human-Computer Interaction* 7, ISS (2023), 217–241.

[26] Forrest Huang, John F Canny, and Jeffrey Nichols. 2019. Swire: Sketch-based user interface retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–10.

[27] Yue Jiang, Luis A. Leiva, Hamed Rezazadegan Tavakoli, Paul R. B. Houssel, Julia Kylmälä, and Antti Oulasvirta. 2023. UEyes: Understanding Visual Saliency across User Interface Types. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–21. doi:10.1145/3544548.3581096

[28] Yue Jiang, Changkong Zhou, Vikas Garg, and Antti Oulasvirta. 2024. Graph4GUI: Graph Neural Networks for Representing Graphical User Interfaces. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[29] Christoph Albert Johns, João Marcelo Evangelista Belo, Anna Maria Feit, Clemens Nylandsted Klokmose, and Ken Pfeuffer. 2023. Towards flexible and

robust user interface adaptations with multiple objectives. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–17.

[30] Christoph Albert Johns, João Marcelo Evangelista Belo, Clemens Nylandsted Klokmose, and Ken Pfeuffer. 2023. Pareto optimal layouts for adaptive mixed reality. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–7.

[31] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R Klemmer, and Jerry O Talton. 2013. Webzeitgeist: design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3083–3092.

[32] Thomas Langerak, Kashyap Todi, Ben Lafreniere, Ruta Desai, and Tanya Jonker. 2026. XAIUI: User belief-driven explainable AI for context-aware adaptive interfaces. *ACM Transactions on Interactive Intelligent Systems* 16, 2 (2026), 1–28.

[33] Geonsun Lee, Min Xia, Nels Numan, Xun Qian, David Li, Yanhe Chen, Achin Kulshrestha, Ishan Chatterjee, Yinda Zhang, Dinesh Manocha, et al. 2025. Sensible Agent: A framework for unobtrusive interaction with proactive AR agents. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*. 1–22.

[34] Chenyi Li, Guande Wu, Gromit Yeuk-Yin Chan, Dishita Gdi Turakhia, Sonia Castelo Quispe, Dong Li, Leslie Welch, Claudio Silva, and Jing Qian. 2025. Satori: Towards Proactive AR Assistant with Belief-Desire-Intention User Modeling. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–24.

[35] Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. 2022. Learning to denoise raw mobile UI layouts for improving datasets at scale. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–13.

[36] Jiahao Nick Li, Yan Xu, Tovi Grossman, Stephanie Santosa, and Michelle Li. 2024. OmniActions: Predicting digital actions in response to real-world multimodal sensory inputs with LLMs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–22.

[37] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, and Brad A Myers. 2021. Screen2vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[38] Zhipeng Li, Yi Fei Cheng, Yukang Yan, and David Lindlbauer. 2024. Predicting the Noticeability of Dynamic Virtual Elements in Virtual Reality. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 954, 17 pages. doi:10.1145/3613904.3642399

[39] Zhipeng Li, Christoph Gebhardt, Yves Inglin, Nicolas Steck, Paul Streli, and Christian Holz. 2024. SituationAdapt: Contextual UI optimization in mixed reality with situation awareness via LLM reasoning. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[40] Zhipeng Li, Christoph Gebhardt, Yi-Chi Liao, and Christian Holz. 2026. Automating UI Optimization through Multi-Agentic Reasoning. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. 1–20.

[41] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[42] David Lindlbauer. 2022. The future of mixed reality is adaptive. *XRDS: Crossroads, The ACM Magazine for Students* 29, 1 (2022), 26–31.

[43] David Lindlbauer, Anna Maria Feit, and Otmar Hilliges. 2019. Context-aware online adaptation of mixed reality interfaces. In *Proceedings of the 32nd annual ACM symposium on user interface software and technology*. 147–160.

[44] Alexander Lingler, Dinara Talypova, Jussi PP Jokinen, Antti Oulasvirta, and Philipp Wintersberger. 2024. Supporting task switching with reinforcement learning. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[45] Lars Lischke, Sven Mayer, Katrin Wolf, Niels Henze, Harald Reiterer, and Albrecht Schmidt. 2016. Screen arrangements and interaction areas for large display work places. In *Proceedings of the 5th ACM International Symposium on Pervasive Displays*. 228–234.

[46] Feiyu Lu, Leonardo Pavanatto, Shakiba Davari, Lei Zhang, Lee Lisle, and Doug A Bowman. 2024. "where Did My Apps Go?" Supporting Scalable and Transition-Aware Access to Everyday Applications in Head-Worn Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* (2024).

[47] Michał Patryk Miazga, Hannah Bussmann, Antti Oulasvirta, and Patrick Ebel. 2026. Log2Motion: Biomechanical Motion Synthesis from Touch Logs. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. 1–22.

[48] Hee-Seung Moon, Yi-Chi Liao, Chenyu Li, Byungjoo Lee, and Antti Oulasvirta. 2024. Real-time 3d target inference via biomechanical simulation. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.

[49] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* (2021).

[50] Abhinav Kumar Rastogi, Blaise Aguera y Arcas, Chongyang Bai, Jindong (JD) Chen, Srinivas Kumar Sunkara, Xiaoxue Zang, and Ying Xu. 2021. UIBert: Learning Generic Multimodal Representations for UI Understanding. In *the 30th International Joint Conference on Artificial Intelligence (IJCAI-21)*. https://arxiv.org/abs/2107.13731

[51] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 3982–3992.

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. ImageNet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[53] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems* 36 (2023), 68539–68551.

[54] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting user interfaces with model-based reinforcement learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–13.

[55] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2018. Familiarisation: Restructuring layouts with visual learning models. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces*. 547–558.

[56] Kashyap Todi and Tanya R Jonker. 2023. A framework for computational design and adaptation of extended reality user interfaces. *arXiv preprint arXiv:2309.04025* (2023).

[57] Stéphane Tufféry. 2011. *Data Mining and Statistics for Decision Making* (1st ed.). Wiley Publishing.

[58] Daniel Vogel and Ravin Balakrishnan. 2004. Interactive public ambient displays: transitioning from implicit to explicit, public to personal, interaction with multiple users. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*. 137–146.

[59] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems* 32 (2019).

[60] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*. 353–355.

[61] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile UI using large language models. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–17.

[62] Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. 2021. Screen2words: Automatic mobile UI summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 498–510.

[63] Jason Wu, Titus Barik, Xiaoyi Zhang, Colin Lea, Jeffrey Nichols, and Jeffrey P Bigham. 2022. Reflow: automatically improving touch interactions in mobile applications through pixel-based refinements. *arXiv preprint arXiv:2207.07712* (2022).

[64] Jason Wu, Yi-Hao Peng, Xin Yue Amanda Li, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. 2024. UIClip: A Data-driven Model for Assessing User Interface Design. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 45, 16 pages. doi:10.1145/3654777.3676408

[65] Jason Wu, Xiaoyi Zhang, Jeff Nichols, and Jeffrey P Bigham. 2021. Screen parsing: Towards reverse engineering of ui models from screenshots. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 470–483.

[66] Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. 2026. ContextAgent: Context-aware proactive LLM agents with open-world sensory perceptions. *Advances in Neural Information Processing Systems* 38 (2026), 167509–167543.

[67] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations (ICLR)*.

[68] Difeng Yu, Ruta Desai, Ting Zhang, Hrvoje Benko, Tanya R Jonker, and Aakar Gupta. 2022. Optimizing the timing of intelligent suggestion in virtual reality. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–20.

[69] Jiayi Eurus Zhang, Joost Broekens, and Jussi PP Jokinen. 2024. Modeling cognitive-affective processes with appraisal and reinforcement learning. *IEEE Transactions on Affective Computing* 16, 2 (2024), 771–782.

A LLM Suggestion Prompts

```
# Initial facet suggestion with zero-shot LLM
SYSTEM: You are an XR (Extended Reality) context-aware functional access support system.
You are given a catalog of facet candidates. Each facet represents a user-facing functionality of an app.
A facet is an intent-level functional unit, typically expressed as a verb + noun pair.
Examples:
- spotify.view_now_playing: checking the currently playing media
- spotify.browse_music: browsing and exploring music
- slack.browse_channels: navigating Slack channels
- slack.browse_messages: reading message content within a channel
Each facet also has capabilities. Capabilities are atomic functional building blocks inside an app, such as APIs, tools, or
minimal interaction units. Some capabilities may have the form open_<facet_name>. This indicates that the current facet can
directly open or navigate to another facet. The catalog may expose these navigation edges explicitly through linked_facets
and whether a facet is_entry_facet.
Facets may also form a facet graph:
- some facets are broader entry or navigation points
- some facets are more specific, deeper functionalities
- "Open Section" is a special entry/root facet representing the app's starting point or navigation hub
Use this graph structure to reason about granularity and navigational usefulness:
- prefer the facet whose granularity best matches the user's likely immediate need
- recommend a broad entry facet when the user likely needs general access or exploration within an app
- recommend a deeper facet when the context suggests a specific functionality directly
- recommend adjacent parent/child facets together only if they serve distinct roles in the scenario
- avoid filling the list with redundant facets that are merely consecutive steps in the same navigation chain
- use open_<facet_name> capabilities and graph links to infer what is reachable from what
- use linked_facets and is_entry_facet to reason about navigation hubs, nearby functionalities, and non-redundant coverage
Your job is to predict a set of 10 functional facets most likely to be useful for the user's current Task and Environment.
Consider the following types of usefulness:
- primary: direct support for the immediate task
- secondary: supportive or peripheral actions that help the task
- ambient: low-attention or background utilities the user may keep around
Selection guidance:
- prioritize facets that would be useful to surface directly in XR
- balance specificity with breadth across likely needs
- do not over-concentrate on a single app unless the context strongly calls for it
- when multiple facets from the same app are plausible, choose the smallest set that gives meaningful access without
unnecessary redundancy
Rules:
- Recommend exactly 10 unique facets whenever the catalog contains at least 10.
- Use only canonical facet_id values from the provided catalog.
- Use the facet graph only to improve relevance and coherence; do not output graph structure.
- If a specific deep facet is recommended, ensure the choice is justified by the task and environment rather than by graph
proximity alone.
Return JSON only: {
"summary": "brief strategy explanation",
"recommendations": [{
"facet_id": "canonical.facet_id",
"tier": "primary|secondary|ambient",
"confidence": 0.0,
"reason": "short explanation"
}]}
```

```
USER:
Current scenario:
Task: {task_id}
Environment: {environment_id}
{details_block}
Available facet catalog:
{catalog_text}
Instructions:
- Predict the {top_k} facets most likely to be useful.
- Cover primary, secondary, and ambient support when plausible.
- Use linked_facets and entry facets when they help explain likely navigation paths or adjacent utilities.
- Use exact facet_id values from the catalog.
- Return JSON only.
```

```
# Initial facet suggestion with few-shot LLM
SYSTEM: You are an XR (Extended Reality) context-aware functional access support system.
You are given:
1. a catalog of facet candidates, where each facet represents a user-facing functionality of an app
2. historical examples selected under a controlled transfer condition
A facet is an intent-level functional unit, typically expressed as a verb + noun pair.
Each facet also has capabilities. Capabilities are atomic functional building blocks.
Some capabilities may have the form open_<facet_name>, indicating that the current facet can directly open or navigate to
another facet.
The catalog may expose these navigation edges explicitly through linked_facets and whether a facet is_entry_facet.
Facets may also form a facet graph:
- some facets are broad entry/navigation points
- some facets are deeper, more specific functionalities
- "Open Section" is a special entry/root facet representing the app's starting point or navigation hub
Use the graph structure to reason about navigational usefulness and granularity:
- prefer the facet whose granularity best matches the likely immediate need
- use entry/root facets when the user likely needs broad app access or exploration
- use deeper facets when the target context suggests a specific action or information need
- avoid redundant recommendations that are only adjacent steps along the same chain unless they serve distinct roles
- use open_<facet_name> capabilities and graph links to infer likely bundles of reachable or complementary functionality
- use linked_facets and is_entry_facet to reason about navigation hubs, nearby functionalities, and non-redundant coverage
Your job is to predict a set of 10 functional facets most likely to be useful for the user's current Task and Environment.
Consider the following types of usefulness:
- primary: direct support for the immediate task
- secondary: supportive or peripheral actions that help the task
- ambient: low-attention or background utilities the user may keep around
Use the historical examples as transfer priors:
- infer stable patterns from the examples
- use them to calibrate likely primary, secondary, and ambient facets
- do not copy examples blindly
- when examples conflict with the current Task and Environment, prioritize the current context
- use the examples to learn likely breadth, specificity, and app/facet combinations
Selection guidance:
- prioritize facets that would be useful to surface directly in XR
- balance specificity with breadth across likely needs
- do not over-concentrate on a single app unless strongly supported by both context and examples
- when multiple facets from the same app are plausible, choose the smallest non-redundant set that gives meaningful access
Rules:
- Recommend exactly 10 unique facets whenever the catalog contains at least 10.
- Use only canonical facet_id values from the provided catalog.
- Treat the examples as weak priors, not ground-truth labels for the target case.
- Use graph structure to improve coherence, not to mechanically reproduce chains from examples.
- If a specific deep facet is recommended, ensure it is supported by the target context or transfer evidence, not by graph
adjacency alone.
Return JSON only: {
"summary": "brief strategy explanation",
"recommendations": [{
"facet_id": "canonical.facet_id",
"tier": "primary|secondary|ambient",
"confidence": 0.0,
"reason": "short explanation"
}]}
```

```
USER:
Current scenario:
Task: {task_id}
Environment: {environment_id}
{details_block}
Transfer condition: {condition.label}
Interpretation: {condition.description}
Historical examples for this condition:
{support_block}
Available facet catalog:
{catalog_text}
Instructions:
- Predict the {top_k} facets most likely to be useful.
- Cover primary, secondary, and ambient support when plausible.
- Use linked_facets and entry facets when they help explain likely navigation paths or adjacent utilities.
- Use exact facet_id values from the catalog.
- Return JSON only.
SUPPORT BLOCK:
Example {index}: scenario={scenario_key}
User: {participant_id}
Task: {task_id}
Environment: {environment_id}
Used facets:
{app_name} / {facet_name}
[{facet_id}]
```

For next-facet suggestion, the same prompts as initial facet suggestion were used, but the following was added to the context:

```
Existing user-placed facets in the XR layout:
[
{idx}. facet_id={facet_id}; app={app_name};
facet_name={facet_name}; relevance={rel}
]
Use these already placed facets as prior context, avoid repeating them, and recommend what should come next for the
remaining needs.
```