Enhancing Tech Creation through Multimodal Context - Report

Viktor Ćurčić

Abstract

In this report, we have analyzed the research conducted on XR technologies and their use in technical environments (hardware and software engineering, UX design and data science). We discussed the pros and cons of these technologies and gave a proposal for an XR prototype in tech creation.

1 Introduction

Extended Reality (XR) technologies are rapidly evolving into environments for professional technical work. Modern headsets now provide high-resolution displays, spatial tracking, hand-gesture recognition, and integrated eye-tracking sensors capable of sampling gaze at high frequencies. At the same time, the workflows of software engineers, hardware designers, UI/UX practitioners, and data scientists are becoming more multimodal and demanding. These parallel developments suggest that XR systems, when combined with gaze-aware interaction models, can serve as powerful platforms for complex creation tasks.

This report presents a research-driven prototype of an XR tool designed to support ”tech creation” across four domains: software development, hardware engineering, interface design, and data analysis. The unique feature of this prototype is the integration of eye-tracking both as a direct input and as an implicit proof. Eye-tracking is not employed only to replace pointing devices; rather, it is leveraged as a behavioral signal that reveals intent, focus, confusion, and exploration patterns [1]. The proposed system uses this information to adapt the XR workspace and trigger relevant assistance.

The design presented here is not an abstract thought experiment. It synthesizes relevant research in gaze-based interaction, XR development environments, immersive analytics, and multimodal AI assistance. The architecture, workflows, and design ideas draw from empirical studies showing that gaze can improve navigation efficiency, inform predictive models, and enhance comprehension in complex coding and design tasks [2, 3]. The prototype is thus grounded in both scientific insight and practical engineering feasibility.

2 Background and Related Work

Eye-tracking has long been used in human–computer interaction research to study attention, expertise, and cognitive load. In XR environments, gaze assumes an even more central role because it is continuously available and aligned with the user’s frame of reference. Prior work demonstrates that explicit gaze-based interaction - such as dwell selection, gaze-based menus, or gaze-driven manipulation - can reduce task time and mitigate the need for controller-based pointing. These results generalize across VR and AR systems, suggesting that gaze serves as a highly expressive mode of interaction.

A second body of work examines implicit uses of gaze. For example, fixation patterns reliably correlate with comprehension difficulties in software debugging tasks [3], and attention-weighted models in code completion have shown large performance improvements when augmented with gaze features [2]. Studies of collaborative VR interfaces also indicate that sharing live gaze cursors improves coordination and reduces ambiguity in debugging and design tasks [4].

The emergence of XR-native development environments, such as immersive IDEs, spatial coding platforms, and VR-based visualization tools, provides a natural stage for integrating gaze. Prototypes like SnapSmith and RiftSketch demonstrated early possibilities for immersive programming, though without eye-tracking [5]. More recent systems incorporate gaze for code selection, auto-scrolling, contextual expansion, and intelligent mode switching. Literature from immersive analytics highlights similar opportunities: gaze can drive filtering, brushing, and exploration of 3D visualizations, revealing sequences of analytical actions that traditional logs cannot capture [6].

Taken together, this research motivates a unified XR tool that positions gaze not as an add-on, but as the central organizing signal for creation workflows.

3 System Architecture

The architecture of the proposed prototype is organized around four subsystems: (1) XR rendering and workspace management, (2) the eye-tracking pipeline, (3) the interaction and context engine, and (4) an AI-augmented backend. Figure 1 illustrates these components and their information flow.

The XR rendering layer, implemented in Unity or Unreal Engine with OpenXR, provides a space where tools from each technical domain can coexist without occlusion. Floating panels display code editors, schematic diagrams, UI mockups, and data dashboards. Users can rearrange, resize, and pin these panels in 3D space. The system maintains consistency across sessions by positioning elements relative to the user's physical orientation.

The eye-tracking subsystem streams continuous gaze rays, pupil dilation data, fixation clusters, and blink events through the headset's SDK (e.g. Tobii XR). Signal processing filters small involuntary eye movement, reduces jitter, and aggregates point samples into meaningful fixation windows. These windows become the fundamental units of interaction.

The interaction and context engine binds gaze events to objects in the environment. For instance, when a ray intersects a code token, the engine attaches semantic metadata from the Language Server Protocol (LSP). When gaze lands on a circuit component, the system retrieves its netlist associations. When gaze falls onto a UI mockup, Figma element identifiers are surfaced. This interpretation of gaze allows downstream modules to act with semantic awareness rather than raw coordinates.

Finally, the AI backend uses gaze-augmented context to generate predictions, suggestions, or adaptive transformations. For example, an LLM-based code assistant may prioritize completions based on the user's fixation history, or visualization recommendations may highlight plots relevant to the region the user repeatedly inspects.

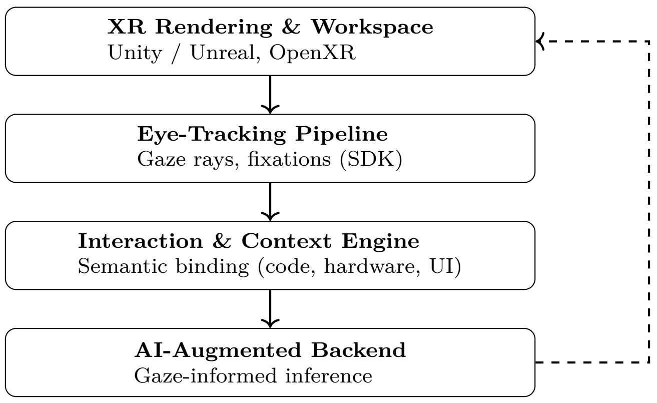
XR Rendering & Workspace
Unity / Unreal, OpenXR
Eye-Tracking Pipeline
Gaze rays, fixations (SDK)
Interaction & Context Engine
Semantic binding (code, hardware, UI)
AI-Augmented Backend
Gaze-informed inference

Figure 1: Compact architecture of the gaze-enabled XR prototype.

4 Eye-Tracking Pipeline and Interaction Model

Unlike conventional input methods, gaze offers both explicit and implicit channels of interaction. The prototype employs a dual-mode model that respects this distinction.

Explicit gaze interaction is reserved for commands that the user intends to execute directly. When the user intentionally fixates on an element - such as a button, a code symbol, or a node - the system interprets this as a potential selection. To avoid accidental activation, dwell thresholds are estimated by the task context; for instance, shorter thresholds are applied during navigation, while longer thresholds are required in editing modes. Visual feedback such as highlight rings or radial progress indicators confirm pending activation.

Implicit gaze interaction, by contrast, continuously interprets the structure of the user's visual scan-path. Long fixations may indicate difficulty, making the system surface documentation or related examples. Rapid alternating eye movement between two regions may signal comparison behavior, causing the system to propose side-by-side views. Extended periods focused on the console may trigger automatic transitions into debugging mode. These behaviors mirror findings in gaze-based modeling of programmer cognition [1].

The gaze pipeline therefore supports a hierarchical interpretation: raw rays feed fixation detection; fixations feed semantic binding; semantic sequences feed machine-learning classifiers.

5 Domain-Specific Design Considerations

5.1 Software Development

In the software development scenario, the XR workspace functions as a spatial IDE composed of panels for source code, terminals, file explorers, documentation, and runtime diagnostics. Eye-tracking simplifies the mechanics of code navigation: users move the text cursor by looking at a location and issuing a small voice confirmation; scrolling occurs smoothly as gaze approaches panel boundaries; definitions or caller graphs appear when the user fixates on a symbol. These features reduce reliance on controller pointing and manual window switching.

Implicit gaze signals serve to enrich AI-assisted development. For example, an LLM may monitor which segments of code the user repeatedly revisits and offer explanations or test generation. If the user's gaze returns to a region after reading documentation, the assistant might infer comprehension verification and suggest relevant refactoring. Such integration aligns closely with recent research demonstrating that gaze-conditioned language models outperform vanilla models in code comprehension tasks [2].

5.2 Hardware Engineering

Hardware design benefits greatly from XR's spatial functionalities. The prototype enables engineers to inspect PCB layouts, mechanical assemblies, and circuit schematics in 3D space. Gaze-based selection allows fine-grained interaction even with densely packed components, since the system maps gaze intersections to EDA metadata rather than relying on pixel-level precision. Fixating on a trace highlights its entire electrical net; dwelling on a component reveals parametric data; examining a region repeatedly prompts the system to generate simulation plots associated with that part of the design.

Eye-tracking also supports early usability evaluation of hardware interfaces. For devices containing screens, buttons, or user controls, gaze heatmaps generated inside the XR mockup reveal attention patterns that traditionally require dedicated eye-tracking laboratories.

5.3 UI/UX Design

The system presents a spatial UI/UX design studio where designers create and evaluate interfaces on large, immersive canvases. Gaze tracking transforms the design process in two ways. First, it serves as a direct manipulation tool: designers select elements simply by looking at them, enabling faster layout iteration. Second, the system acts as an integrated usability testing environment. As designers interact with prototypes, their gaze traces are visualized as heatmaps overlaid on the interface. These heatmaps reveal overlooked components or insufficient contrast. This immediate feedback loop accelerates early design refinement.

5.4 Data Science

Immersive analytics stands to benefit particularly from gaze-driven interaction. Complex datasets can be visualized across multiple floating canvases, allowing analysts to focus naturally on regions of interest. When the user gazes at a cluster in a scatterplot, the system magnifies it, attaches statistical summaries, and proposes related projections or dimensionality-reduction views. Gaze paths indirectly document the analytic reasoning process, enabling the system to construct reproducible traces. Over time, these traces serve as training data for models that learn to anticipate relevant transformations.

6 Prototype Workflow

A typical workflow with the prototype unfolds as follows. The user puts on the headset and performs a brief eye-tracking calibration. The XR workspace loads the most recent arrangement of panels. As the user begins a task, their gaze continuously orients the system: code panels under focus grow sharper; peripheral panels dim slightly to reduce cognitive load.

When the user inspects a complex code segment, the system logs the fixation and retrieves relevant documentation in the background. If the user lingers, a subtle prompt appears offering explanations. Later, when debugging, quick gaze transitions between the console and a suspect function prompt the system to open a comparison view without manual intervention.

During hardware work, looking at a component highlights its electrical net; repeated attention to a thermal hotspot triggers simulation overlays. While designing interfaces, the designer receives real-time feedback about the interface's important areas. In data analysis, gaze-driven brushing selects clusters for deeper inspection.

Throughout all tasks, gaze and system state are logged into a unified dataset for offline study and for training predictive models that refine the prototype's adaptability.

7 Implementation Technologies

Building this prototype requires coordinated contributions from multiple technical domains. The software implementation would revolve around Unity or Unreal Engine, C# interaction scripts, OpenXR device abstractions, and an eye-tracking SDK such as Tobii XR. Semantic services - including code parsing and static analysis - are provided by Language Server Protocol backends. Visualization layers employ WebGL-based frameworks integrated into XR canvases.

On the hardware side, the XR headset must support accurate eye tracking, low-latency rendering, and stable calibration. UI/UX design contributes spatial interaction patterns, gaze-aware widgets, and mockups created in Figma or similar tools. Data science contributes the logging pipeline, gaze-sequence modeling, classification of interaction modes, and attention-weighted recommendation models.

8 Future Work

The prototype presented here establishes a conceptual foundation for gaze-driven XR development tools, but several research challenges remain. First, long-duration XR use raises concerns that must be systematically studied. Second, gaze-based models of developer cognition require careful validation across users and tasks. Third, integrating LLMs into XR workflows introduces issues of latency, context management, and trust. Finally, evaluation methodologies are needed to compare gaze-adaptive XR tools against desktop environments.

Despite these challenges, the convergence of eye tracking, XR interfaces, and multimodal AI suggests that the future of technical creation will be increasingly spatial, perceptually grounded, and adaptive. The prototype described here offers one path toward realizing that vision.

Thank you for your time reading this report. The use of first person plural is strictly a stylistic choice and the paper is of my own personal making. Please contact me with any and all recommendations and criticisms you might have.

References

[1] Naveen Sendhilnathan et al. "Implicit Gaze Research for XR Systems". In: *Reality Labs Research, Meta Platforms* (2024).

[2] Yasmine Elfares, Gül Çalikli, and Mohamed Khamis. "GazeCopilot: Evaluating Novel Gaze-Informed Prompting for AI-Supported Code Comprehension and Readability". In: *arXiv preprint arXiv:2511.08177* (2025). URL: https://doi.org/10.48550/arXiv.2511.08177.

[3] Thomas Weber, Rafael Vinicius Mourão Thiel, and Sven Mayer. "Supporting Software Developers Through a Gaze-Based Adaptive IDE". In: *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023.

[4] Thammathip Piumsomboon, Gun Lee, and Mark Billinghurst. "CoVAR: a collaborative virtual and augmented reality system for remote collaboration". In: IEEE, 2017, pp. 1–10.

[5] Joe Durbin. *Real-Time Coding in Virtual Reality with RiftSketch*. UploadVR. 2016. URL: https://www.uploadvr.com/riftsketch/.

[6] Maxime Cordeil, Tim Dwyer, and Karsten Klein. "Immersive Analytics". In: IEEE, pp. 1–8.