ESP-JETA
ESP Journal of Engineering & Technology Advancements
ISSN: 2583-2646 / Volume 6 Issue 3 July 2026
Paper Id: JETA-V6I3P109/ Page No: 73-83

Review Article

AI-Assisted Design-to-Code Automation for Enterprise Frontend Engineering

Mohammed Siddiq Hussain Nisar Khan
Trine University-Detroit Campus

Received Date: 10 August 2026 Revised Date: 15 August 2026 Accepted Date: 20 August 2026

Abstract: *The success of the latest developments in multimodal artificial intelligence and large language models has made Design-to-Code automation even more efficient and straightforward for converting a written design document into working front-end application source code. With the current explosion of multimodal AI and the advent of generative large language models, the above-mentioned Design-to-Code automation initiative takes on new momentum by turning text design docs into usable application front-end code. While there has been significant progress, such as visual fidelity and code generation accuracy, it is important to recognize that existing literature largely concentrates on maintaining visual fidelity and code generation accuracy with little attention given to software engineering requirements of real-world enterprise systems, such as Enterprise Design System compliance, continuous integration, security, or maintainability requirements. It is a comprehensive overview of the current research in creating novel automatic drawing techniques based on code, such as new developments in rule-based drawing, Deep Learning approaches to drawing, as well as Vision-Language models, Multimodal LLMs and Agentic AI. It's a methodology that has been developed based on experience with the approach PRISMA - Frontend Engineering in the Enterprise Application - which has been created to identify the available approaches, evaluation criteria, application scenarios and deployment challenges. Issues identified in the review for further work are the need for benchmarking, based on enterprise principles; governance mechanisms on board; support for reusable component architectures; and consideration of software quality over the long term. To address the above limitations, a conceptual framework called Enterprise Design-to-Code Automation Framework is proposed that addresses the drawbacks of AI-generated code automation, design systems, quality assurance, governance and DevOps principles.*

Keywords: *AI-Assisted Design-to-Code, Enterprise Frontend Engineering, Large Language Models (LLMs), Multimodal AI, Software Engineering Automation*

I. INTRODUCTION

Now, in the era of software engineering, Artificial Intelligence has changed the whole scenario by automating several complex things during software engineering, like introducing SW source code, testing SW, documentation, debugging, capturing SW requirements and even maintaining SW. One of those is a cool new way to ease the development of software that is between UI/UX design and the implementation in the front-end, which is called Design to Code programming. Throughout history, lots of time and lots of energy have been spent on creating a tool, such as Figma, Sketch, Adobe XD or anything else, and getting it implemented; there's the potential for a lot of inconsistencies, implementation errors and, at best, misunderstandings between the designer and developer in this process [1]. Utilizing deep learning, computer vision, the principles of design, and multimodal LLMs, AI can unravel the organization of design elements, the design pattern, understand the interactions between the various elements of a Graphical User Interface, and generate code ready to use on various platforms including design frameworks like React, Angular, Vue.js, Flutter, or HTML/CSS [2]. Algorithms utilized by smart systems could be designed to favor UI object recognition for high visual accuracy, meaning of the objects in the image, contextual learning, or even code-aware generation to retain the relations of the layout as well as to create reusable code. Some new techniques like Retrieval-Augmented Generation (RAG), AI agents, iterative reasoning and engineering, and self-reflection frameworks have considerably enhanced the ability of D2C systems to generate components based on a given context, validate the generated code automatically, tune the results to design systems and continuously improve the components generated [3]. For this reason, the enterprise believes AI-powered D2C automation plays a major role in the enterprise's digital transformation, lowering software development costs, speeding up software releases, increasing developer productivity, and enhancing design work and collaboration. Now, though, Design-to-Code has grown out of static code and is adaptable: responsive design, accessible code testing, cross-platform support, reusability of design pieces, automatic testing and integration into any type of CI, etc. A time of exploring and experimenting with future proof-of-concept projects has come and gone, and the time for AI-powered front-end automation has come, where it is a strategic component in the software development ecosystem, contributing to the goal of "making scalable, maintainable and intelligent software systems [4]. MultiModal AI, enterprise design systems, software engineering best practices, along with

automated QA, mark another milestone in the smart front-end engineering route and represent a new frontier in autonomous software development environments where developers can leverage AI to create quality digital experiences without adding to the manual workload or optimizing developer productivity [5].

These are amazing advancements; however, implementing AI-powered Design-to-Code technologies into enterprise software engineering presents several technical, organizational, and methodological hurdles that must be addressed if AI is to make an impact on enterprise software engineering [6]. Most of the existing studies have focused on the accuracy of the visual reconstruction or the similarity of pixels or a “cut and paste” type assessment of software engineering qualities like readability, scalability, maintainability, evolution history of the software, modularity, accessibility and security [7]. Additionally, many state-of-the-art D2C models are trained and validated on simple or limited datasets, representing only simple or limited applications of localized components, designs or layout, just too small or limited to represent the complexity of enterprise software applications that feature reusable design systems, component libraries, dynamic business workflows, responsive layout, localization, authentication, etc., and domain-specific user interfaces. Even the existing evaluation approaches fail to offer a common framework of some kind to measure production readiness, the quality of the software, technical debt and sustainability throughout the software lifespan in various frontend settings [8]. However, this has led to a number of issues with the introduction of multimodal LLMs: hallucinated code generation, different component architectures, design drift, clashes on imports and exports, import/export issues and concerns regarding access and security, and enterprise governance policies, all of which raise issues that need to be addressed. In the industrial field, the use of some of the commercial AI tools, such as GitHub Copilot, Vo from Vercel, Figma AI, Cursor and other Intelligent Coding Assistants, is gradually eroding into industrial processes. The principles and ways of validation are low-explored topics in academic research, as are the software quality consequences and their long-term fate, maintainability [8]. Moreover, previous efforts looking at the implications of AI for code generation are mainly either dedicated to broader studies on code generation, neural program synthesis or software engineering automation, or fail to discuss more specific problems related to engineering front-end applications in enterprise settings and to AI-assisted software design application tasks such as ‘Design-to-Code' [9]. There are various crucial challenges that are normally addressed in one way, such as design-system governance, human-AI synergy, DevOps integration, continuous testing, explainability, software compliance and enterprise lifecycle management [10]. Few studies have been conducted; however, in order to systematically integrate all the capabilities of the ‘multimodal AI’ with the principles from the engineering discipline, how can we go forward and package up the research needs into something meaningful for enterprises?

The goal of this review is to leverage motivation to discuss the challenges being faced in research from the project's perspective and to make a comprehensive and systematic overview of automation using AI in the Design-to-Code method from the enterprise frontend engineering point of view. Unlike previous studies that investigated the progress of the algorithms or the overall approach to using AI for code production in general, this study creates an extensive overview of Design-to-Code production-oriented ecosystems. In addition, the present review critically examines the state of the art of the benchmarking approach and provides an overview of some issues still to be addressed from a technical perspective, problems at the enterprise level, and new research horizons in the preprocessing or per-iteration benchmarking approach. The conceptual framework in this paper suggests merging all four aspects of multispectral code generation by AI-enterprise design systems, software quality assurance and governance, continuous integration/continuous deployment (CI/CD) and enterprise code lifecycle management, in enterprise design systems. There is not only code generation, but code generation as a process. EDCAF has been designed that way, in fact, and a much more sustainable, scalable, secure and maintainable code generation of the front end can be part of enterprise software engineering. The proposed system further provides a template for future research, which includes the needs of explainable AI, self-correcting programming, enterprise-grade assessment tests, coordinated development of AI and individual people, and intelligent quality assurance and autonomous frontend engineering. The intent of this review is not to merely summarize the state of the art, but also to provide some tips on how to best leverage the new developers, software architects, front-end developers and software practitioners.

To summarize, the following are the key elements of this Review:

- Conducts a thorough literature review on the most recent advancements in AI-based design-to-code tools.
- Presents a taxonomy of Design-to-Code approaches from the perspective of various types.
- Using a critical but appropriate perspective, explain the new challenges and opportunities in the context of software maintainability, software governance, software accessibility, software security, software adherence to design systems and feedback.
- Proposed a novel automation framework for a scalable and Reliable Webfrontend Engineering process by extracting web design systems' code from software QC and DevOps pipelines.
- New concepts in the development of new-generation enterprise software within the upcoming Strategic Research include autonomous agentic frontend engineering, standardized evaluation benchmarks for enterprise and XAI.

II. RELATED WORK

With the latest developments in technology, Design-to-Code (D2C) has grown to become a much more streamlined engineering process as opposed to a manual and time-consuming process. Recent developments in AI have taken D2C from a manual to a more streamlined engineering process. In the early design-to-code systems, most of these systems were utilized to transform all of the predefined graphic elements into fundamental HTML/CSS coding. There were no OO designs, except for the majority of designs, which were rule-based and template-based, which meant you could take this bunch of elements for graphics and produce static HTML/CSS. These resulted in easy-to-use interfaces without any semantic awareness, flexibility or extensibility for the needs of complex enterprise applications [8]. Convolutional Neural Networks (CNNs) can now take advantage of deep learning techniques to become an encoder-decoder process and could even be applied to ground-breaking systems like Pix2Code and, more recently, neural code generation networks to automatically identify the different elements of a user interface such as buttons, icons and text from screen images and even wireframes [9]. Extensive long-range spatial dependency and context-dependency between the different elements of interfaces are captured by the newly developed architecture, such as Transformer-based architectures and ViTs, which has resulted in the perception of complex layouts, leading to the improvement of structural accuracy and code quality [10]. A second revolution in the field has now begun: VLMs and MLLMs on the same architectures, with their input as images, natural language reasoning, and even source code, taken directly from a design model & a text specification, even an entire Frontend implementation [11]. That's one whereby systems have to be designed to be more complex systems, and another has to be there systems of system design inclusion and access, system design coding standards specifications/includes, and systems for ensuring contextual consistency and reduction of errors from a system design contextual area during development of systems [12]. While software engineering collaboration or some form of collaborative approach is virtually universal in these approaches, they are also iterative in nature, generating, validating, testing, optimizing and further enhancing interpretations, codes, and other aspects, agentic AI approaches are completely new and gaining in popularity with multiple agents adding value in interpreting, generating, validating, testing, optimizing and continuing to enhance performance [13]. The existing literature mainly has disadvantages, most of the works obtained from the literature have been developed based on a visual similarity measure, but in an enterprise, there are other factors that are also important but not considered in the obtained works, such as software maintainability, software accessibility, software security, software design system compliance, software technological debt, software governance, software sustainability during software lifecycle and others [14]. Besides, the majority of the benchmark datasets are only designed for simple web pages, which fail to consider reusability of architectures, responsiveness, micro-frontend, localization and an industrial frontend application pipeline of a continuous integration environment [15]. There are significant developments in automated UI generation which are documented in the literature, but they have been scattered across the computer vision and computer engineering communities, human–computer interaction and generative AI communities. The latter should be an extensive synthesis, both aggregating these points of view as well as highlighting the needs in enterprise-frontend engineering and its development for future research and use of trustworthy DTC automation systems in an enterprise [16].

Table 1: Summary of Recent Literature on AI-Assisted Design-to-Code Automation

Technique / Approach	Primary Contribution	Strengths	Research Limitations
Rule-Based Design-to-Code Systems [8]	Introduced template-driven conversion of graphical user interface elements into HTML/CSS, demonstrating the feasibility of automated frontend generation.	Simple implementation, deterministic output, low computational requirements, suitable for structured interfaces.	Limited semantic understanding, poor adaptability to complex layouts, lacks scalability for enterprise applications, and cannot generalize to unseen designs.
Deep Learning-Based UI Code Generation (CNN, Encoder-Decoder) [9]	Applied convolutional neural networks and sequence-to-sequence architectures for automatic recognition of UI components and code synthesis from screenshots.	Improved visual recognition, enhanced layout detection, and higher code generation accuracy than rule-based approaches.	Performance degrades for complex enterprise interfaces, limited contextual reasoning, and weak support for reusable component architectures.

Vision Transformer (ViT)-Based Models [10]	Utilized transformer architectures to capture long-range spatial dependencies and hierarchical relationships among interface components.	Better structural understanding, improved layout consistency, and superior feature representation for complex interfaces.	High computational complexity, extensive training requirements, and limited evaluation in industrial frontend environments.
Vision-Language Models (VLMs) [11]	Combined visual understanding with natural language reasoning to generate frontend code directly from design artifacts and textual specifications.	Multimodal reasoning, better semantic interpretation, supports multiple frontend frameworks, and improved contextual understanding.	Susceptible to hallucinations, inconsistent component mapping, and insufficient evaluation of maintainability and software quality.
Multimodal Large Language Models (MLLMs) [12]	Leveraged multimodal foundation models to integrate screenshots, design files, textual prompts, and programming knowledge for end-to-end frontend generation.	High-quality code generation, strong contextual reasoning, multi-framework support, and improved developer productivity.	Limited enterprise benchmarking, challenges in accessibility compliance, security validation, and lifecycle maintenance.
Retrieval-Augmented Generation (RAG)-Based Systems [13]	Enhanced code generation by retrieving enterprise design systems, reusable components, coding standards, and project documentation during inference.	Reduces hallucinations, improves design-system consistency, increases component reuse, and supports enterprise coding standards.	Retrieval quality depends on knowledge base completeness, introduces additional computational overhead, and lacks standardized evaluation metrics.
Agentic AI Frameworks [14]	Introduced collaborative AI agents responsible for design interpretation, code generation, testing, debugging, optimization, and iterative refinement.	Supports autonomous software engineering workflows, continuous improvement, collaborative reasoning, and higher production readiness.	High architectural complexity, increased computational cost, limited industrial deployment, and absence of standardized evaluation protocols.
Enterprise Design-to-Code Platforms [15]	Investigated the integration of AI-assisted frontend generation with design systems, DevOps pipelines, and enterprise software engineering workflows.	Promotes scalable development, continuous integration, governance, and collaboration between designers and developers.	Existing studies primarily emphasize workflow integration while providing limited discussion of software quality metrics and long-term maintainability.
Systematic Reviews on AI-Assisted Frontend Engineering [16]	Reviewed recent advances in AI-assisted frontend development, software automation, and intelligent code generation while identifying emerging research trends.	Provides a comprehensive understanding of current technologies, identifies research challenges, and highlights future opportunities.	Lacks a unified enterprise-centric taxonomy integrating multimodal AI, Design-to-Code automation, governance, DevOps, software quality assurance, and lifecycle management.

III. REVIEW METHODOLOGY

To ensure that a literature selection is conducted in a transparent, rigorous and reproducible way, the methodology was followed in the review, in accordance with the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) approach. Summarize and/or go through the up-to-date advancements in the space of Design-to-Code (D2C) automation from an AI standpoint, and recognize, screen, evaluate and analyze the level of scholarly literature. A thorough attempt was made through the scientific databases, identifying the papers published from the year 2020 to 2026 in all the scientific databases such as Scopus, Web of Science, IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect and Google Scholar. The approach to exploring how to use keywords to be searched: Design-to-Code, AI-assisted frontend development, Large Language Models, Vision-Language Models, Multi-modal AI & UI Code generation, Software engineering automation, and retrieval-augmented generation and agentic AI. The old way included splitting the records, screening the

title/abstract, and lastly splitting the eligibility of the record full text based on a number of predetermined criteria. The papers (according to the paper criteria), the high-quality and peer-reviewed papers, influential papers, the technically valuable and the benchmark papers from conferences were prioritized. In addition, methodology, the study area, use of AI tools, study evaluation and measurement of the study limits were assessed in the study. After aggregating all the evidence collected from the multiple sources, a detailed taxonomy was built, which facilitated the identification of the actual research lines, gaps and identified research areas, and also the foundations of the proposed Enterprise Design-to-Code Automation Framework.

Inclusion Criteria

- Research papers on the topic of AI-assisted Design-to-Code automation, Frontend engineering or Frontend AI Smart Code Generation that have been released from 2020 to 2026.
- All papers from Digital Libraries.
- Research papers that showcase the use and/or recreation of cutting-edge Artificial Intelligence technologies, including, but not limited to, core topics like deep learning, Vision-Language Models, Large Language Models and agentic technologies such as Retrieval-Augmented Generation.
- Evaluation of Experimental data, Comparative Studies, Enterprise Applications, Benchmark Datasets/software engineering implications of published work.
- Exclusion Criteria
- Publications published before 2020, such as editorials, tutorials, blogs, white papers & patents, dissertations, non-peer-reviewed works.
- Other projects that are executed on DTWay, Frontend, SW Eng., Multi-Modal and Intelligent Coding.
- Once a year, an annual international conference for young (Papers) bioengineers.
- Do NOT repeat published work of others (unless quoted in a few sentences). Display or do research.

IV. EVOLUTION OF DESIGN-TO-CODE AUTOMATION

In software engineering, Design-to-Code automation has come a long way from its rule-based and deterministic roots to a more multimodal and intelligent automation that now has the power to make decisions on the frontend, without human intervention [17]. The paradigm needed to be recoded using hand-made rules and static HTML/CSS for the graphical user interface elements mapped to heuristics, which made the D2C interface inefficient; that, coupled with many other serious flaws of the paradigm, meant that manual recoding was necessary. Despite showing that the generation of UIs was feasible with these systems, they were not able to reason about the intended semantics; adapt to unexpected changes; or scale to the context of today's enterprise software [18]. The breakthroughs in deep learning and CNNs and encoder-decoder models came a long way in helping users understand components given screenshots and wireframes, and consequently, to generate precise frontend code in structured programs. Later, Transformers and ViTs enhanced the understanding of layout by considering inter-relations and long-range spatial relations between items within an interface. Later, layout models have

been developed that enhanced the exploitation of interrelations and long-range spatial relations between the content items using their interface architecture [19]. However, with the emergence of Vision-Language Models and Multimodal Large Language Models, a substantial breakthrough was made, integrating models for visual data and natural language understanding into a comprehensive tool capable of generating actual production-ready applications based on both design files and text requirements [20]. Recently, Retrieval-Augmented Generation (RAG) functionality has been introduced, allowing models to pull enterprise design systems, libraries of reusable components, and organizational coding conventions into their code generation process, thus minimizing hallucinations and improving consistency [21]. The latest is the idea of agentic AI frameworks where a group of independent agents collaborate to analyze design, generate code, validate, test, optimize and refine through iterations [22].

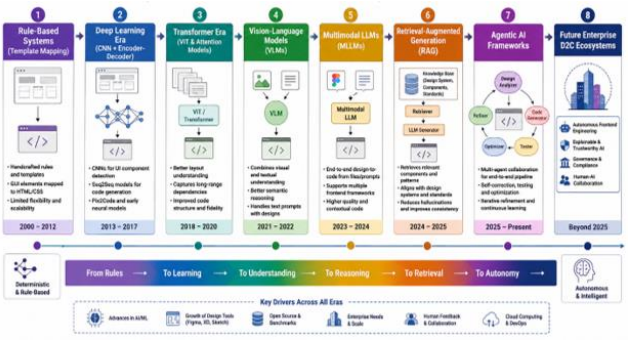

Figure 1: Evolution of AI-Assisted Design-to-Code Automation

V. PROPOSED CONCEPTUAL FRAMEWORK

Multimodal Artificial Intelligence (AI), Large Language Models (LLMs), Vision-Language Models (VLMs) and agentic AI have greatly enhanced the ability of Design-to-Code (D2C) automation. However, most of the existing frameworks and best practices are limited to creating a visually accurate front-end production code and fail to consider enterprise software engineering principles like maintainability, scalability, accessibility, governance, security, design system compliance and continuous software evolution. Also, existing design-to-code pipelines are often created as noninterdependent code generation models that don't support other aspects of the enterprise software development lifecycle. The constraints imposed by these are overcome by presenting a conceptualization of multimodal integration with Enterprise frontend design-to-code practices like Enterprise Design-to-Code Automation Framework (EDCAF). In addition to the "Design-to-Code Automation" model, EDCAF is multi-layered to enable intelligent design interpretation, contextual reasoning, Enterprise knowledge retrieval and automated code synthesis, Quality assurance, Governance, DevOps integration and continuous learning. The first element of the framework is collecting and capturing design artifacts in different formats such as Figma files, wireframes, screenshots, Design Tokens, Natural language requirements, etc. These inputs are used by multimodal perception modules that allow the structure of the interface to be learned, the semantic dependencies between multimodal information, and learning of the user's interactions. Next, reasoning modules can combine with LLM support, VLMs, Retrieval-Augmented Generation (RAG), and enterprise knowledge bases to generate a collection of front-end components with the consistency of re/ordering terms in specific contexts, coding rules, and enterprise design systems. Through Continuous Integration/Continuous Deployment (CI/CD) pipelines, automated validation modules are defined and execute pre-deployment accessibility, performance, maintainability, security, visual consistency and responsiveness testing of created apps. Finally, feedback and production telemetry from humans are introduced on a layer of iterative learning, enabling models and knowledge about design in the enterprise to be iterated over time. EDCAF allows you to transform your prototype design work into an engineered enterprise system, with automation. The potential framework therefore provides a comprehensive journey to creating trustworthy solutions as an end-to-end process with a view to users and making these solutions reliable, interpretable, scalable and sustainable, ready to assist with digital transformation projects in the future.

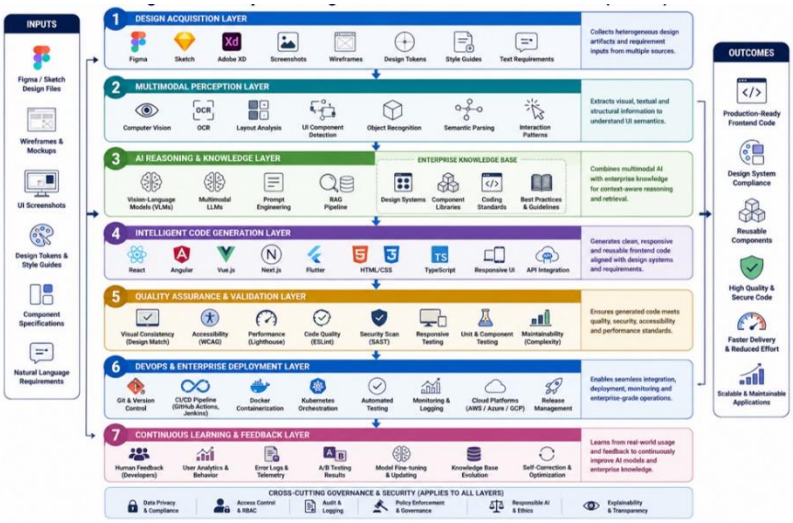

Figure 2: Enterprise Design-to-Code Automation Framework (EDCAF)

The proposed EDCAF stresses that enterprise Design-to-Code automation is not a code generation process, but a software engineering process. The current systems, however, only address the functionality of generating code on the frontend, whereas EDCAF adds enterprise-specific features-knowledge-based reasoning and retrieval-augmented generation, automated code quality checking, powerful governance enforcement and continuous improvement during the entire lifecycle. This multi-layered design ensures that once deployed by the developed applications, they are following the enterprise design patterns, coding standards, security controls, and accessibility controls. Furthermore, a feedback-based learning mechanism that enables continuous improvement of the model by feeding developer corrections, monitoring the data that is collected during the production cycle, analyzing the interactions of users and feeding the data from software maintenance is also possible. Closed loop - work on the quality of code, end the technical debt, avoid hallucinating issues and optimise the organization's knowledge reuse. Therefore, EDCAF can provide a way to easily integrate into the front-end engineering designs of enterprise software programs, and leave behind explainable and trustworthy software engineering practice that is sustainable. It's also very low-level and can be replaced with future innovations like self-healing user interfaces, autonomous software agents, explainable code generation or, in enterprise design systems, a semi-sentient or self-generated system, and makes EDCAF a good reference guide for future generations of front-end creation.

Table 2: AI Techniques Incorporated in the Proposed EDCAF

AI Technique	Role in EDCAF	Primary Function	Advantages	Limitations
Computer Vision (CV)	Design Acquisition & Perception	Detects UI elements, icons, text, layout structures, and visual hierarchies from screenshots and design files.	Accurate visual parsing and component localization.	Sensitive to complex or noisy interface designs.
Optical Character Recognition (OCR)	Multimodal Perception	Extracts textual information from design artifacts including labels, menus, and form fields.	Preserves textual semantics and improves interface understanding.	Performance may decrease with stylized or low-resolution text.
Convolutional Neural Networks (CNNs)	UI Feature Extraction	Learns visual representations of interface components for object recognition and layout analysis.	Efficient feature extraction and mature implementation.	Limited capability to model long-range contextual relationships.
Vision Transformers (ViTs)	Layout Understanding	Captures global spatial dependencies and structural relationships among interface components.	Superior layout comprehension and contextual reasoning.	Computationally intensive and requires large training datasets.
Vision-Language Models (VLMs)	Multimodal Understanding	Integrates visual content with textual descriptions to infer design intent and interface semantics.	Enables multimodal reasoning and better semantic understanding.	May generate ambiguous interpretations for complex interfaces.
Multimodal Large Language Models (MLLMs)	Intelligent Code Generation	Generates frontend code from combined visual designs and natural language specifications.	Produces high-quality, context-aware, multi-framework implementations.	Risk of hallucinations and inconsistent component generation.
Retrieval-Augmented Generation (RAG)	Knowledge Integration	Retrieves enterprise design systems, coding guidelines, reusable components, and documentation during generation.	Improves consistency, component reuse, and reduces hallucinations.	Effectiveness depends on the quality and completeness of the knowledge repository.
Prompt Engineering	AI Reasoning	Optimizes AI interactions through structured prompts and task-specific instructions.	Enhances generation accuracy and controllability.	Requires iterative refinement and domain expertise.
Agentic AI Frameworks	Workflow Orchestration	Coordinates multiple AI agents for design analysis, code generation, testing, debugging, and optimization.	Supports autonomous software engineering workflows and iterative refinement.	High architectural complexity and increased computational overhead.
Reinforcement Learning from Feedback (RLHF)	Continuous Learning	Continuously improves model behavior using developer feedback, production analytics, and validation results.	Enables adaptive learning, improved software quality, and long-term performance.	Requires high-quality feedback data and continuous model maintenance.

VI. ENTERPRISE FRONTEND ENGINEERING

Enterprise frontend engineering isn't just about automating code for the user interface; it's about building applications that are scalable, maintainable, secure, and compliant with governance and can function in complex software environments. However, enterprise systems cannot be built on any single architecture; as longstanding systems are strategy-

focused on translating a visual design into code, they need to have capabilities in design systems, DevOps, QA and governance, and need to be end-to-end designed. Common best practices for frontend development in such contexts are the creation of mini-classrooms of libraries for reusables, micro-frontend designs, responsive web development techniques, adopting CI/CD pipelines, and establishing a standard code. This means that the frontend engineering has to not only produce accurate images, but it must also abide by enterprise design tokens, accessibility Guidelines, security protocols, and browsing performance, additionally providing long-term observability and maintenance capabilities. The first step of the enterprise architecture is the extraction of all the different design elements, such as design requirements, screenshots, wireframes, design files from Figma, and design tokens, etc. It is based on a new proposed Enterprise Design-to-Code Automation Framework (EDCAF). They then go through multimodal perception models where both Computer Vision (CV), Optical Character Recognition (OCR), and Vision Transformers based on ViT are included in order to identify the interface components and to understand their semantics in the layout. The code generation intelligence modules then generate highly performant frontend production interfaces for the frameworks. The software created goes through automatic verification: accessibility, security, performance and design-system compliance before deployment. Finally, DevOps pipelines are automated for version control, tests, packaging with containers, deployment, production monitoring and governance, and continuous feedback from the developer and production environment are fused together through Reinforcement Learning from Human Feedback and agentic AI. This seamless design ensures that the organization can move easily from a code-generation-based AI model to a smarter overall enterprise system software engineering evolution that learns and adapts to all aspects of software quality and compliance with regulatory standards and other factors that guarantee operational reliability.

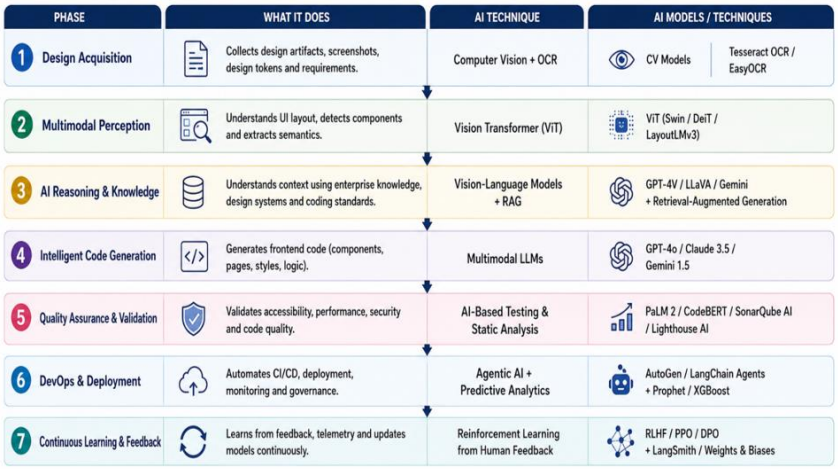

Figure 3: Enterprise AI-Assisted Frontend Engineering Architecture Based on EDCAF

VII. LIMITATIONS AND CHALLENGES

Artificial Intelligence (AI) has made significant strides in the realm of Design-to-Code (D2C) automation, but there are still some technical challenges that have prevented it from being widely used in enterprise front-end engineering. AI has been making some significant strides with Design-to-Code (D2C) automation, but there are some challenges in the technology that haven't yet been addressed before it can be adopted for enterprise front-end engineering. While there are several good candidates for the generation of software using tools, most of these current tools are geared towards copy/paste fidelity and syntactic correctness, and less toward producing actual software for production that would address issues like maintainability, scalability, design modules, security, or long-term evolution of the codebase that are studies and concerns in enterprise engineering. LLMs, given multimodal inputs, often produce hallucinated outputs, incorrect flow of information, styling inconsistencies, repeated code and redundant dependencies, and a lot of manual effort involved in getting rid of them prior to deployment. Moreover, AI-built apps frequently do not adhere completely to the enterprise design systems, accessibility regulations (WCAG), responsive design rules, and corporate coding customs. One of the major challenges is a lack of business logic context. Though visual systems can be a pretty good reconstruction from available Design-to-Code systems, the ones that are essential for enterprise applications involving application workflows, state control techniques such as AJAX, API interactions, authentication methods, and the expected dynamic behaviors of the user end up being quite difficult to deduce. Another drawback of current models is their lack of ability to generalize, due to the absence of a uniform dataset for large-scale industrial applications. But most governance data sets are of a single small webpage or just a screenshot from a single UI and do not support the many scenarios of the real-world enterprise, which includes reusable component libraries, micro fronts, localization, supporting different versions of software, complex software architecture, etc. Additionally, energy usage, infrastructure needs, latency, and power consumption of multimodal systems and transformer-

based models are significant hurdles to real-time deployment for resource-constrained organisations. Two elements showed an improvement with the introduction of LLM (large language model) powered tools such as Retrieval-Augmented Generation (RAG) models and agentic AI, and they are the enterprise knowledge repository and contextual reasoning. Therefore, dependable, understandable, secure, and production-quality code generation in frontends is a fundamental research dilemma that should be addressed in the near future, and the central one in this research area, which includes multimodal reasoning, intelligent validation, automated testing, and Enterprise-aware software engineering methodologies.

Apart from the technical ones, there are several organizational, governance, ethical, and operational issues that must be addressed to automate Design-to-Code at scale in enterprises through the use of AI. The biggest challenge is the lack of a unified evaluation framework that can rate the production readiness of software for various software quality aspects, such as maintainability, accessibility, performance, security, explainability, technical debt, and sustainability of its lifecycle. The existing assessment processes are mostly about whether the reconstruction was done well or how close the parts are to the actual object and do not provide much durability in terms of software quality and reliability in its use. Addressing accountability, licensing, privacy, and regulatory matters is equally crucial in the context of platforms and their developed software. Deliberation over governance issues, like accountability, licensing, privacy, and rules, ought to go into software. The danger lies in data confidentiality breaches, unauthorized disclosure of knowledge, and cybersecurity concerns in enterprise applications with support for front-end development coupled with access to sensitive company information, customer data, and proprietary design assets. Additionally, many organizations have well-established workflows for software development, as well as legacy systems and software engineering practices, which implies the seamless integration of AI-generated code into existing DevOps pipelines is an organizational challenge. People are also fundamental, and software designers may not wish to work with software they can't understand, explain, and verify. Empty bell rings are responded to, new decision responsibly issues are raised, as is the case for autonomous AI, new questions arise to answer like who is responsible for a decision, how is it supposed to be monitored, what happens when there is a model shift, how is it possible to prevent bias being passed from generation to generation, how can it be governed when more than one organization is involved? Explainable AI, automated quality assurance, continuous human oversight, and learning and feedback mechanisms are key for future enterprise ecosystems and will not supplant humans, but rather augment and support them. To overcome all these challenges, interdisciplinary studies and research are needed to develop and create reliable, scalable and sustainable solutions between design and coding that can be deployed to facilitate the next generation of intelligent FEA.

VIII. FUTURE RESEARCH DIRECTIONS

According to the next generation of AI-driven Design-to-Code (D2C) automation, it's not just code generation models that will be isolated, but actually autonomous, self-reasoning, self-planning, self-evaluating, and self-upgrading software engineering ecosystems. Future work is suggested to focus on systems that integrate multiple specialized agents that collaboratively execute each part of the design interpretation, requirement analysis, component recommendation, code generation, debugging, testing, optimization, and documentation process. These intelligent agents are not about to develop static front-end code; they are meant to work with each other throughout the Software Development Lifecycle (SDLC) to get enterprise-compliant applications that are maintainable and reusable. Research emphasizing the enhancements of other multimodal foundation models that understand visual designs, textual specifications, user stories, API specifications, business workflows, and organizational design systems is another promising research avenue. These models should include Retrieval-Augmented Generation (RAG), graph-based knowledge representation, as well as domain-specific reasoning to construct architectures for the frontends consistent with the coding practices of enterprise support. Other directions to explore in future work include controlling mechanisms for models to learn on their own, reinforcement learning, and continual learning mechanisms that enable the models to evolve with developer feedback and production system data such as software maintenance logs. Besides, there is still a lack of research in the creation of standardized enterprise scale bench mark datasets that represent realistic software engineering scenarios covering the reusability of component libraries, localization and accessibility issues, responsive interface and cross-platform deployment etc. micro-frontends. Trust will also be reinforced by advancing XAI, causal reasoning, and uncertainty estimation, which help to provide transparency in the explanations of the code and architectural decisions that systems produce.

There is also a need for further studies in building enterprise-wide engineering frameworks that seamlessly integrate the Design-to-Code code generation automation along with software governance, DevOps practices, cybersecurity, and long-term lifecycle management. Current models primarily focus on producing correct code; future systems must be able to improve more than one software quality attribute at a time, decreasing technical debt, adhering to software regulations, saving energy resources, optimizing accessibility, boosting security, and improving software maintainability. The deployment of researched AI-based frameworks to automatically validate developed applications with organizational architecture frameworks, accessibility standards (WCAG), security policy (OWASP), coding standards, and enterprise regulations should be studied. Bringing the automation of Design-to-Code to fruition through cloud-native architectures,

CI/CD pipelines, Infrastructure-as-Code (IaC), and intelligent software observability platforms for creating fully automated frontend engineering workflows is another emerging topic. Collaborative environments between humans also present the opportunity to build systems that allow developers, designers, and AI agents to engage in an iterative process of interaction and decision-making in the construction of software. Research on managing AI should explore methodologies for creating frameworks and processes of fairness, transparency, accountability, protection of intellectual property rights, privacy, and compliance with regulations for software developed using tools. Standard evaluation frameworks and maturity models are being developed, specifically for enterprise Design-to-Code systems, which will further facilitate comparisons and industrial uptake. Last but not least, integration, adaptation, and cross-disciplinary efforts among researchers from AI, software engineers, human-computer interaction, cybersecurity, and enterprise architecture will be necessary to build trustworthy, explainable, resilient, and sustainable AI-enabled frontend engineering ecosystems to power the digital transformation journeys of today's enterprise organizations with quality software and operational reliability.

IX. CONCLUSION

With the influence of AI, D2C automation- once a prototype domain- now gives a greater possibility for the paradigm of enterprise frontend engineering in a short time. This is a systematic review that has looked at the evolution of D2C technologies from rule-based models to DL models, followed by VLMs, MLLMs, RAG, and agentic models. Surprisingly, although the recent models on Software Engineering (SE) have reached great advancements focusing on visual fidelity and functional correctness, software engineering practices in enterprise software, like maintainability, scalability, usability, securing software and its development lifecycle, security, and governance, are not being widely researched for AI. Furthermore, there are some weaknesses that should be improved, such as: No standards to assess enterprise quality; Limited support for reusable component architecture; Hallucination in generated code; Limited standards application to design systems; and poor adherence to DevOps and continuous application delivery pipelines. To solve these drawbacks, the multimodal needs of the enterprise problem areas were taken into consideration in the proposition of the multimodal EDCAF: A conceptual layered architecture that integrates multimodal perception, intelligent reasoning, enterprise knowledge retrieval, automated code generation, quality assurance, DevOps deployment, and continuous learning into an integrated enterprise engineering workflow. Furthermore, it offered a taxonomy of techniques and future directions for research from the perspective of the enterprise, such as autonomous software development, explainable AI, common performance benchmarks, human-AI interaction and co-working, intelligent quality assurance, and responsible governance. Overall, this is a detailed and comprehensive study of development and provides a practical way forward for building trustworthy, scalable, and production-ready AI-frontend-engineering systems. The results are expected to direct researchers, software architects, and industry professionals to better implement next-generation enterprise software development with intelligent, design-to-code automation.

- **Interest Conflicts:** The author(s) declare(s) that there is no conflict of interest concerning the publishing of this paper
- **Funding Statement:** Not Applicable

X. REFERENCES

[1] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., ... & Xie, X. (2024). A survey on evaluation of large language models. ACM transactions on intelligent systems and technology, 15(3), 1-45.

[2] Beltramelli, T. (2018, June). pix2code: Generating code from a graphical user interface screenshot. In Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems (pp. 1-6).

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[4] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

[5] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In International Conference on Machine Learning (pp. 8748-8763). PmLR.

[6] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in neural information processing systems, 33, 9459-9474.

[7] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & McGrew, B. (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

[8] Si, C., Zhang, Y., Li, R., Yang, Z., Liu, R., & Yang, D. (2024). Design2code: Benchmarking multimodal code generation for automated front-end engineering. arXiv preprint arXiv:2403.03163.

[9] Cho, J., Kang, D., Kim, H., & Lee, G. G. (2025). Self-Correcting Code Generation Using Small Language Models. arXiv preprint arXiv:2505.23060.

[10] Li, Z., Wu, X., Du, H., Liu, F., Nghiem, H., & Shi, G. (2025). A survey of state-of-the-art large vision-language models: Benchmark evaluations and challenges. In Proceedings of the Computer Vision and Pattern Recognition Conference (pp. 1587-1606).

[11] Gülmez, B. (2026). Code generation with large language models: a survey from neural program synthesis to autonomous software

development. Applied Intelligence, 56(6), 200.
[12] Tao, Y., Li, Y., Qin, Y., & Liu, Y. (2025). Retrieval-augmented code generation: A survey with focus on repository-level approaches. arXiv preprint arXiv:2510.04905.
[13] Acharya, M., Zhang, Y., Leach, K., & Huang, Y. (2025). Optimizing code runtime performance through context-aware retrieval-augmented generation. arXiv preprint arXiv:2501.16692.
[14] Lv, Z., Poiesi, F., Dong, Q., Lloret, J., & Song, H. (2022). Deep learning for intelligent human–computer interaction. Applied Sciences, 12(22), 11457.
[15] Ganesaraja, R., AP, S., Rathinasamy, K., Amancharla, C., Das, R., Panse, S. D., ... & Vijayakumar, S. (2025). Beyond Prototyping: Autonomous, Enterprise-Grade Frontend Development from Pixel to Production via a Specialized Multi-Agent Framework. arXiv preprint arXiv:2512.06046.
[16] Geruslu, V., Aliyeva, Z., & Tüzün, E. (2026). Factors Influencing the Quality of AI-Generated Code: A Synthesis of Empirical Evidence. arXiv preprint arXiv:2603.25146.
[17] Jain, V., Agrawal, P., Banga, S., Kapoor, R., & Gulyani, S. (2019). Sketch2Code: transformation of sketches to UI in real-time using deep neural network. arXiv preprint arXiv:1910.08930.
[18] Chen, J., Chen, C., Xing, Z., Xia, X., Zhu, L., Grundy, J., & Wang, J. (2020). Wireframe-based UI design search through image autoencoder. ACM Transactions on Software Engineering and Methodology (TOSEM), 29(3), 1-31.
[19] Jiang, Y., Zheng, Y., Wan, Y., Han, J., Wang, Q., Lyu, M. R., & Yue, X. (2025). Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. arXiv preprint arXiv:2507.22827.
[20] Yue, C., Chai, J., Zhang, Y., Ding, Z., Liang, X., Wang, P., ... & Lin, W. (2025, November). UIOrchestra: Generating High-Fidelity Code from UI Designs with a Multi-agent System. In Findings of the Association for Computational Linguistics: EMNLP 2025 (pp. 2769-2782).
[21] Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., ... & Neubig, G. (2025, May). Openhands: An open platform for AI software developers as generalist agents. In International Conference on Learning Representations (Vol. 2025, pp. 65882-65919).
[22] Yang, Z., Hong, W., Xu, M., Fan, X., Wang, W., Cheng, J., ... & Tang, J. (2025). UI2Code^ N: A Visual Language Model for Test-Time Scalable Interactive UI-to-Code Generation. arXiv preprint arXiv:2511.08195.