Applied AI Letters

WILEY

PERSPECTIVE OPEN ACCESS

From Idea to Implementation: Evaluating the Influence of Large Language Models in Software Development—An Opinion Paper

Sargam Yadav[1] | Asifa Mehmood Qureshi[1] | Abhishek Kaushik[1] | Shubham Sharma[2] | Roisin Loughran[1] | Subramaniam Kazhuparambil[3] | Andrew Shaw[1] | Mohammed Sabry[4] | Niamh St John Lynch[1] | Nikhil Singh[5] | Padraic O'Hara[1] | Pranay Jaiswal[1] | Roshan Chandru[1] | David Lillis[6]

[1]School of Informatics and Creative Arts, Dundalk Institute of Technology, Dundalk, Ireland | [2]The Centre for Research in Engineering Surface Technology (CREST), TU Dublin, Dublin, Ireland | [3]Zendesk, Dublin, Ireland | [4]ADAPT Centre, Dublin, Ireland | [5]National College of Ireland, Dublin, Ireland | [6]School of Computer Science, University College Dublin (UCD), Dublin, Ireland

Correspondence: Abhishek Kaushik (abhishek.kaushik@dkit.ie)

Received: 23 November 2025 | **Revised:** 7 May 2026 | **Accepted:** 18 June 2026

Keywords: ChatGPT | coding | large language models | natural language generation | software development | transformer

ABSTRACT

The introduction of transformer architecture was a turning point in natural language processing (NLP). Models based on the transformer architecture, such as bidirectional encoder representations from transformers (BERT) and generative pretrained transformer (GPT), have gained widespread popularity in various applications such as software development and education. The availability of large language models (LLMs) such as ChatGPT and Bard to the general public has showcased the tremendous potential of these models and encouraged their integration into various domains such as software development, for tasks such as code generation, debugging, and documentation generation. In this study, opinions from 11 experts regarding their experience with LLMs for software development have been gathered and analyzed to draw insights that can guide successful and responsible integration. The overall opinion of the experts is positive, with the experts identifying advantages such as an increase in productivity and reduced coding time. Potential concerns and challenges such as risk of overdependence and ethical considerations have also been highlighted.

1 | Introduction

The swift evolution of artificial intelligence (AI) in the past decade has allowed advanced natural language processing (NLP) tools to perform tasks such as translation, summarization, and information retrieval. These tools are becoming more easily available and integrate seamlessly with our daily lives in the form of virtual assistants. Large language models (LLMs) are large and complex deep learning models with millions or billions of parameters trained on vast amounts of data. Models such as ChatGPT can carry out human-like conversations keeping context in mind, highlighting their advanced natural language understanding (NLU) [1] and natural language generation (NLG) [2] capabilities. Transformer-based models such as the bidirectional encoder representations from transformers (BERT) [3] and generative pretrained transformer (GPT) [4] have demonstrated unparalleled capabilities in tasks such as text generation [5], named entity recognition [6], machine translation [7], text classification [8–11], and more. LLMs are pretrained on large amounts of data and can gain a considerable amount of domain knowledge, which allows them to be a valuable assistant in fields such as software development [12], healthcare [13], education [14, 15], and agriculture [16–19]. Since their launch, LLMs such as GPT-4 [20], Codex [21], and CodeBERT [22], have been utilized by researchers and practitioners to efficiently perform

various tasks in the software development workflow such as code completion [23], code synthesis from natural language [24], debugging, code review (CR) [25], code comment and documentation generation [26], among others.

In the context of software development, LLMs can greatly reduce workload for developers, allowing them to focus on more creative aspects of the coding process [27]. They can streamline the development process by ensuring compliance and version control, improving code quality, and promoting collaboration [28]. They can also be leveraged to develop interactive educational tools that can provide tailored feedback to programming students [29, 30]. Moreover, recent advances in agentic AI mark a significant leap in software development tools, driven by their ability to set complex goals in dynamic, unpredictable environments and to carry out tasks autonomously [31]. Table 1 displays a list of the commonly used LLMs for software development, along with their parameters, training data, training language, and architecture.

However, there have also been certain risks and limitations associated with using LLMs. LLMs may not fully understand the context of a prompt [33], leading to erroneous responses. Overreliance on LLMs to perform tasks may inhibit problem-solving and critical thinking skills. There are also several ethical considerations of using LLMs for software development, such as unintended biases [34], security risks [35], transparency, and explainability [36]. LLMs are trained on vast amounts of data, which contains human-like biases [37]. The models can reflect these biases in their responses and cause further harm. It is important to mitigate biases and address the concerns to ensure fairness in the model and its related outputs.

In this study, a group of software development experts, consisting of researchers and practitioners, was surveyed to understand the potential applications of LLMs in software development. Analysis of opinions obtained through a loosely structured written interview from 11 experts, consisting of researchers and practitioners, has also been performed to draw insights about the potential of LLMs. These opinions were gathered in 2023; therefore, discussion on Agentic AI and the latest tools is out of scope of this study. The rest of the article is structured as follows: In Section 2, we discuss the motivation behind conducting this survey, as well as define the hypothesis and research questions. Section 3 details the methodology used in the study, Section 4 describes the opinions of experts regarding the integration of LLMs in processes such as code generation, review, debugging, and so on. Section 5 provides an overview and analysis of the expert opinions, as well as a discussion of the research questions. Section 7 concludes the review study.

2 | Motivation

LLMs such as GPT, CodeBERT, and Codex have demonstrated transformative potential in software development. The hype surrounding LLMs creates ambiguity regarding their potential applications and drawbacks. There is some concern about job security, as it is perceived by some that models will become better at performing software development tasks than professional developers [38]. Although LLMs can be an invaluable tool for learning, students may also use them unfairly and compromise academic integrity [39, 40]. Additionally, as with any new technology, there may be a steep learning curve before it can be seamlessly integrated [41]. In order to get a clear picture of the benefits and drawbacks of LLMs in the field of software development, it is important to analyse the opinions of experts working in the field. The motivation behind conducting this study is to analyse and perform thematic coding on written interviews obtained from 11 experts, including researchers and practitioners, in the area of machine learning and AI and about the use of LLMs in software development.

The hypothesis of the survey is as follows:

Hypothesis. *Experts such as researchers and practitioners have a positive outlook toward the integration of LLMs in software development.*

The research questions structured to explore the hypothesis are as follows:

1. RQ1: What are the applications and advantages of LLMs in software development as per expert opinions?
2. RQ2: What are the limitations and potential concerns of LLMs in software development as per expert opinions?

TABLE 1 | Large language models for software development.

Model	Parameters	Training data	Language	Architecture
CodeParrot	1.5b	GitHub	Python	GPT-2
GPT-Neo	2.7b	Pile dataset [32]	—	GPT-2 based
GPT-NeoX	20b	Pile dataset	—	GPT-3
GPT-J	6b	Pile dataset	—	GPT-3
Codex	12b	GitHub, +	Python, +	GPT-3
CodeBERT	125M	GitHub	Multiple	BERT
CuBERT	—	GitHub	Python	BERT
PolyCoder	2.7b	GitHub	12 languages	GPT-2

3 | Methodology

The study employed loosely structured written interviews to obtain open-ended, experience-driven opinions from experts in software development, applied AI, and other related fields. This approach was selected to allow participants to articulate nuanced opinions and perspectives in their own words, which is consistent with the paper's aim of reporting and synthesizing expert opinion rather than conducting a formal qualitative investigation through thematic analysis.

The study has been performed by obtaining loosely-structured written non-risk interviews from experts, which include practitioners and researchers. Practitioners are individuals who are currently working with software development in industry, and researchers are working on LLMs and other related areas. The search for the experts was done through Google Scholar, by contacting fellow researchers in the laboratory, and scheduling meetings with acquaintances. 36 experts, 18 males and 18 females, were contacted across the globe to get their opinions on the use of LLMs in software development. The study contains the opinions of the 11 responses that were received. The experts are not familiar with each other, and therefore, group bias is avoided. Consent has been obtained from the experts to summarize and share their opinions and reduce overlapping content. The interviews were non-risk and did not require the disclosure of any personal information from the interviewer.

Responses were reviewed iteratively by the authors to identify recurring themes and topics, advantages and drawbacks of the technology, and points of disagreement across expert opinions. These patterns have been synthesized and discussed in the results and analysis section. Given the opinion-oriented scope of the study, the analysis has been performed through manual interpretive synthesis rather than formal qualitative thematic coding.

Table 2 displays the gender and role information for the experts interviewed in the study. There are a total of three female and eight male experts. There are four practitioners and seven researchers. Out of the male experts, two are practitioners, and six are researchers. Out of the female experts, two are practitioners, and one is a researcher. Thematic coding of the opinions is performed by two annotators, and inter-annotator agreement is calculated using Cohen's κ in Section 5.

TABLE 2 | List of the experts with their roles and gender.

No.	Role	Gender
Expert 1	Practitioner	Male
Expert 2	Practitioner	Female
Expert 3	Researcher	Male
Expert 4	Researcher	Male
Expert 5	Researcher	Female
Expert 6	Researcher	Male
Expert 7	Researcher	Male
Expert 8	Researcher	Male
Expert 9	Practitioner	Female
Expert 10	Researcher	Male
Expert 11	Practitioner	Male

4 | Expert Opinions

In this section, the opinions of the experts have been analyzed and categorized based on different aspects such as the role of LLMs in code generation and review, NLU, quality assurance, predictive analysis, developer collaboration, as well as bias and ethical considerations of LLMs for code generation. The experts were given these subheadings as a prompt, and some responded to one or more of them. The content from the unstructured essays was classified into the mentioned classes without any modification or rephrasing.

4.1 | The Role of LLMs in Code Generation

LLMs are extensively used in code generation activities. These language models have the capability to generate accurate and efficient code. This section discusses the opinions of the experts on utilizing LLMs in code generation tasks.

Expert 1: The advent of LLMs has rapidly accelerated software development. Getting started with a new library, new programming language, or a new tech stack in general has never been easier. A smart use of LLMs employed by our team is translating code across different languages. As a team of Scala Engineers, we had to develop a new glue service in Pythonor and for some of the nuances, we wrote Scala code and translated it into Python using LLMs. While there is ease of use, the obvious caveat when using such models is that the burden of proof is on the user. However, in the case of software development, the accuracy of such models can be easily and fairly validated as if the code generated is correct, there should not be any compilation errors. Moreover, unit, split, and integration tests should validate that the business logic is preserved.

Expert 4: To illustrate the effectiveness of LLMs in code generation, a few case studies and examples are examined. GitHub Copilot [27], powered by OpenAI's Codex, is an example of LLMs in Code generation. Using Copilot, developers can automatically generate code based on their comments in natural language. It can even be corrected in real-time by entering and modifying the text, which significantly speeds up code writing and adds more efficiency. LLMs have also been used to translate code between different programming languages [42]. Language and cross-tool problems can be overcome using source code translation integration. For example, if the code snippet is written in English, it can be transformed into Python, Java, or any other language. This facilitates cross-language collaboration and code reuse, particularly in diverse development environments. However, LLMs' potential role in code generation is part of the picture, and some issues and obstacles must be addressed. Communication is an innate human skill, but when people talk to one another, sometimes the language they use is unclear, such that it has more than one interpretation. It is almost impossible to implement such instructions in a program. In some cases, LLMs might not be

able to grasp the user's ambiguous instructions, causing them to generate [21] code that is not desired. Since LLMs take data from a wide range of texts on which they are trained, they may not always include highly specialized or domain-specific programming concepts. This problem hinders LLMs' ability to generate code that follows specific standards or advanced programming.

Expert 8: The tasks like code generation, creating summaries, and solving problems are easily handled by LLMs. They are mainly used in code development to automatically generate complex code. This can save developers time and effort. LLMs has ability to understand different programming languages and can produce accurate code snippets. This helps developers by improving productivity and enhancing code quality, which will help in reducing errors. By this, developers can focus on high-level design and problem-solving tasks with more efficient development processes [43]. LLMs such as Codex and GPT-4 show good abilities in summarizing, creating, and understanding code, and they can solve complex programming problems effectively. They efficiently address intricate programming issues with a notable success rate. LLMs can also help in speeding up the development process using features like code completion. LLMs can make coding more efficient by reorganizing it and can work directly in development environments to provide real-time help and support by analyzing code. In addition to this, LLMs can be customized for specific projects, which makes them versatile tools in software development. The code generation is made much easier by automating and streamlining the coding process through advanced models like Codex. Evaluating these models' capability to produce correct and efficient code snippets from prompts shows that these AI tools have the potential to change software development, making coding tasks faster and easier for developers [21]. Also, [44] presents case studies showing the effective use of LLMs for producing control code from piping and instrumentation diagrams in an industrial setting. The study uses reference piping and instrumentation diagrams to test the LLMs' ability to identify controllers and generate control code. The LLMs were able to generate plausible interlock code and startup sequences despite challenges like misinterpretation of graphical elements and misaligned signal references. It is found that the generated code required adjustments for practical use, highlighting the need for more contextual information in the prompts to improve accuracy. The other case study focuses on control logic generation for a butane regeneration system. This study uses piping and instrumentation diagrams. This method faced challenges with hallucinated controllers and inaccuracies in identifying equipment. But it is found that by focusing on a specific diagram section and refining prompts, it successfully generated syntactically correct structured text code for piping and instrumentation diagrams, control loops, and startup sequences. This case study illustrated the importance of detailed prompts and the potential for LLMs to produce functional code when given concrete operational parameters and corrected for identified errors.

Expert 10: Other emerging approaches to augmenting and improving LLMs may also find applications in code generation, such as retrieval augmented generation (RAG) and chain of thought to improve code generation [45]. RAG allows more verifiable and predictive results by searching a data source to prime the prompt with additional data. Mixed models, which route to a collection of more specialized models are being developed, for instance, Mixtral [46]. The result of many of these developments is that models are now getting better at the same time as getting smaller and faster to train. The recent release of Gemini 1.5 shows significant advancement in multi-mode models, integrating video, image and, language. Tools are already coming to market that use AI to convert design directly to code, without the need for any conventional coding. A recent example of this is Builder.io, which takes web designs in Figma documents and converts them directly to code, producing HTML, React, and JavaScript just as a developer would [47]. Another unfolding space is the generation of No Code AI Application Development. These tools allow a user to generate business solutions by integrating AI models [48]. An example of this tool is Imagica, which provides a graphical interface that allows users to specify and configure AI models to meet business objectives without the need for any traditional code. There are other tools that allow the rapid integration of LLMs to build solutions, such as Langchain. Agents are another area of growth that has the potential to replace many tasks currently performed by code, or even replace the activity of developers themselves. The emergence of open source models has also significantly lowered the barriers to LLM model development, as demonstrated by Stanford with the Alpaca model they trained from Llama at low cost [49]. This model also employed generative AI to produce training data. This opens up the possibility for companies to develop their specialized models, which would produce a whole new area of software development and related tools. A further result of open-source models is the interest in running local LLMs. Not only can companies develop their own proprietary models, but they can also run on their own infrastructure. Beyond that, this looks set to power AI at the Edge as the next generation of processors should be able to run these models in mobile, desktop, and IoT devices. Along with the growth of tools for development, there is likely to be a significant evolution in AI tools for operations. This will include security tools, Infrastructure as Code, and improved integration and deployment tools. In addition, the tools necessary for training and deploying LLMs will require new solutions. AI Augmented Testing and Security tools have evolved rapidly alongside LLMs. These tools automate tasks that were previously manual, expensive, and required high levels of expertise. These tools also enhance the throughput of the continuous integration (CI)/continuous development (CD) pipeline, which will be essential as AI code generation is likely to increase the velocity and volume of new products. Indeed, every part of the software supply chain management should be enhanced to support productivity increases and the growth of new projects that were previously not commercially viable.

4.2 | Enhancing NLU for Developers

LLMs use prompts that enhance communication and understanding of natural language, thus providing leverage to developers. Expert opinions that discuss the application of LLMs in enhancing NLU are discussed below:

Expert 1: Companies like Amazon (Code Whisperer), Microsoft (through their collaboration with OpenAI and ChatGPT),

Google, and Facebook (through their models Bard and Llama [Open Source], respectively) are active players in this space. There is a strong focus and investment toward enhancing developer experience and productivity.

As LLMs get integrated with scrum software such as JIRA and Google Docs, they can be used for writing tasks such as documentation and planning sprints, without human errors, which can greatly benefit teams. Furthermore, such tools can greatly accelerate the onboarding of new engineers. For example, a new engineer can ask the LLM to explain a line of code or doc in the context of the entire org without having to ping individual engineers or get lost in the Confluence rabbit hole.

Expert 4: Models like GPT-3 and BERT use advanced deep-learning approaches to comprehend and create text resembling human speech. This capability has been applied to transform casual user interaction in natural languages into working code snippets [1]. For example, a programmer can describe any function or algorithm in plain English, and the language model can easily change this description into a function or algorithm in any coding language required. The potential of bridging the gap between natural language and code helps developers to work with higher efficiency and productivity. Projects like GitHub Copilot demonstrate the potential of LLMs in real-world applications. Nevertheless, the questions that involve natural language ambiguity and the persistence of doubts about domain knowledge concerning applying the LLMs in code generation should be addressed.

4.3 | LLMs in CR and Quality Assurance

This section highlights the application of LLM in CR and quality assurance in the opinion of experts.

Expert 1: Add-ons such as GitHub Co-Pilot start being adopted at the enterprize level, and human-escaped bugs and quality assurance gaps can be filled with the model running additional checks to ensure business logic and data integrity/coherence.

Expert 2: CRs are essential in reducing risks arising from medical devices in development and once placed on the market. The introduction of LLMs into the CR process is now an urgent necessity to ensure Quality Assurance is adequately supported and capable of identifying and removing potentially life-threatening risks to patients [50]. Moreover, the potential benefits of introducing LLM for improvement in effectiveness and accuracy of the CR performed with impact on the early development lifecycle are significant, including: (a) best practice in coding standards readily available using automated tools; (b) continual and automated consistency check of coding style and practices; (c) automated commenting of code by the LLM, freeing up the coder to deal with complex tasks; (d) automation of CR for vulnerability against known vulnerabilities and smells; (e) removal of waste (human resources, time, and cost) from manual reviews and improving the effectiveness of the review; (f) enabling integration of static and dynamic code analysis with suggestions from the LLM for action by developers; (g) promotion of learning and competency amongst coders; (h) avoidance of overreliance on quality or validation functions to find the bugs; and (i) lastly, self-learned adjustments from deep learning with developer feedback mechanisms improve the performance of the LLM within an organization, so that "skills" are not lost to a competitor from resignation or retirement [51, 52]. Also, the use of LLMs for CR is a necessity in a risk-adverse, highly complex area of software development, and with the ever-increasing software-related defects identified in medical devices, including recalls and adverse events. Madera and Tomoń [52] demonstrate the benefits of a prediction model for CRs with a high precision of 97% and an accuracy of 92% for large-scale medical software projects considered worthy of implementation. The ability to demonstrate measurable improvements from the introduction of LLM in CR is self-evident in terms of project performance and, most importantly, the potential for mitigation of clinical and patient risk.

Expert 8: LLMs help developers understand code by summarizing it, finding and fixing bugs, correcting errors, and LLMs can also help in speeding up the development process using features like code completion. LLMs can make coding more efficient by reorganizing it and can work directly in development environments to provide real-time help and support by analyzing code.

Expert 11: CR is a vital part of software development. CR allows reviewers to identify defects and provide feedback, allowing developers to address the issues. Typically, the CR process consists of two steps: defect identification and repair. Zhao et al. [25] highlight various LLM models that can be used for CRs, namely ChatGPT, InCoder, CodeT5+, CodeFuse, LLaMA, CodeGen-2, CodeLLaMA, and CodeReviewer. All of these models can understand both natural language and programming languages, and they can also produce superior code snippets. Once the model is finalized, the second step is identifying the prompts. Prompts are input strings sent to a language model to specify the desired response [53]. Prompts are the primary means by which users interact with LLMs. Prompt design has a significant impact on inferences drawn, so it is important to consider when using LLMs for specific tasks. It is critical to identify the optimal prompt that maximizes performance while reducing implementation effort. According to Zhao et al. [25], the following prompts can be used for these models.

"Fix the following buggy code snippet...."

"Fix the following buggy code snippet according to the suggestion in the '//<Comment>' line."

As per Johnson [54], LLM offers the following advantages in the CR process.

1. *Efficiency*: LLMs can review code faster and more accurately than human reviewers. This is especially important in large codebases, where manually reviewing every line of code can take a long time.
2. *Consistency*: Unlike humans, LLMs do not tire or lose concentration. They can consistently provide high-quality reviews, regardless of code volume or complexity.
3. *Continuous learning*: As LLMs review additional code, they learn and grow

This continuous learning process helps them improve their CR skills over time.

4.4 | Collaborative Software Development With LLMs

The utilization of LLMs provides a platform for sharing and discussing ideas, reviewing code, and fostering a collaborative development environment. This section focuses on the collaborative aspects of LLMs from experts' perspectives.

Expert 1: At scale, CI/CD workflows can greatly benefit from LLMs as they can create efficient testing methods via automatically generating mock data and associated unit, split, and integration tests just by feeding business logic and code to the model. Further, LLMs can set version control system standards across a company by enforcing descriptive commit messages and commit history. This can ease collaboration in large-scale organizations where many devs are working on the same code base in parallel. Project management and estimation can be vastly improved by feeding objectives and key results (OKRs), financial targets, engineering and product capacity, and market and competition trends.

Expert 6: One of the impactful techniques in software development is pair programming, which is where two programmers work together at one workstation. One programmer writes code, while the other reviews each line of code as it's typed in, providing feedback and thinking ahead strategically. Pair programming improves code quality, reduces development cycle time, and accelerates the feedback loop. The recent advancements in large neural networks, particularly those built on transformer-based architectures, have revolutionized code generation. These advancements have enabled the integration of such models into pair programming workflows. GitHub Copilot stands out as one of the most mature models in this domain, facilitating collaborative coding with its sophisticated capabilities. Also, in a large-scale survey conducted by the GitHub Copilot team, they examined the benefits experienced by developers using GitHub Copilot. Key findings include:

- *Fast coding*: Developers who used GitHub Copilot [27] completed the task significantly faster–55% faster than the developers who didn't use GitHub Copilot.
- Enhanced developer satisfaction: 60%–75% of users reported increased job fulfillment, reduced frustration while coding, and the ability to focus on more fulfilling tasks.
- Conservation of mental energy: Users noted that GitHub Copilot helped maintain workflow (73%) and saved mental effort during repetitive tasks (87%), contributing to overall developer happiness. This is crucial, as context switches and interruptions can negatively impact productivity.

4.5 | LLMs and Predictive Analysis in Software Development

LLMs are trained on a vast amount of data that helps in various aspects by providing predictions by analyzing historical data. The applications of LLMs for predictive analysis from the viewpoint of experts are given below:

Expert 1: Code completion based on the overall code-base context can be accelerated rather than just syntax completion. Based on company-wide knowledge, OKRs can be set using these LLMs.

4.6 | Bias in LLMs for Software Development

As LLMs are trained on a large amount of internet data, bias in their output is inevitable. This section discusses the biases present in different LLMs and how they can impact the results.

Expert 5: LLMs are generally trained on the large text data available on the internet. This data is largely human-generated or collected through systems created by humans [55]. Therefore, AI bias, also known as machine learning bias or algorithm bias, describes AI systems that generate biased outcomes that mirror and reinforce human prejudices in a community, including social inequity that exists today and has existed historically. The original training set, the algorithm, or the predictions the algorithm generates can all contain bias. It can be inherited in the form of gender stereotypes, religion, and racial prejudices [56]. When producing text, the models can pick up on and replicate these biases that can result in discrimination in automated decision-making systems, among other negative outcomes, in addition to reinforcing negative perceptions [57]. For example, GPT-3, which OpenAI released in 2020, showed an unparalleled capacity to produce coherent and contextually relevant text through training on a massive corpus of text data [36]. Different studies reveal that content generated by GPT-3 is biased in many ways. A study investigates and reports brilliance bias in two models of GPT-3 [58]. Also, another study findings show that some gender preconceptions in generated short stories that appear to be benign are reproduced by GPT-3 [59]. Similarly, the code generated by these LLMs can also produce unfair results, especially in handling bias-sensitive tasks. Due to the increasing popularity of LLMs in software development, these biases may have far-reaching effects, including discriminatory hiring practices, lending choices in the financial sector and biased medical care. For example, when used in the recruitment process, the candidate's personal image and behavior are just as important as their level of professionalism. It is challenging for language models to accomplish this, whether it is over video chat or another form of communication. It requires specific analytical abilities, such as evaluating the tone of voice, gaze, and facial expressions of the candidate's response to ascertain whether the candidate is qualified for the position. Therefore, in order to minimize bias when using AI for recruiting, it also needs to be continuously monitored and the outcomes regularly analyzed by the relevant parties. In the event of issues, this could be more expensive than using conventional hiring techniques [60]. Moreover, the preconceptions in the natural languages are reflected in the codes generated by these models if the task involves any protected attributes, that is, age, education, race, gender, city, region, and so on. For example, potential bias is detected in code snippets generated by five different state-of-the-art LLMs to assess employability, given

the attributes [34]. Also, studies investigating the replacement of human feedback with LLMs report verbosity bias, that is, the systems prefer long or more wordy answers than humans, even if they appear of lower quality [61]. It can also lead to unnecessarily complex and lengthy code generation for software development. Therefore, the software developed by these codes would also reflect bias in its results that can lead to unfair treatment of a special group, culture or region.

Expert 9: LLMs have the power to dramatically influence and improve our lives. As the use of LLMs becomes more prevalent, behaviors, we must remain vigilant to ensure we properly assess all ethical considerations, particularly when these systems are applied within safety-critical domains, such as healthcare. There have been many reported incidents in recent years of LLMs producing biased or discriminatory results. Therefore, we must be extra vigilant in their use, particularly when they are deployed in situations where they could potentially cause harm. It is well documented that medicine has been developed for the white, male patient [62] and unfortunately, AI systems have the power to exacerbate this problem [63]. If we accept the world as it is, with all its flaws, AI systems such as LLMs will inevitably exhibit such behaviors and it is the most vulnerable patients in our society who will suffer the consequences. One of the fundamental, most popular uses of LLMs is as a chatbot—a mechanism for the user to engage with the model in a query-response interaction. The instant, and seemingly intelligent, responses an LLM can offer to individual queries has the potential to resemble a human conversation. While we know this is not true human interaction, it can appear genuine enough to simulate human communication. As such, these techniques have been proposed for use as healthcare chatbots, including some applications such as mental-health chatbots [64]. There are clear potential benefits to the use of such technology for this type of application, for instance, a chatbot is always available, a chatbot does not get tired, a chatbot will never judge you. Users may find the non-judgmental, always-available support to be of great help, particularly if they feel stressed at unsociable times such as during the night or in response to stressful situations where they do not have the opportunity to arrange to speak to a person. However, a chatbot is not a trained professional, and there is a danger in allowing it too much influence over a person's mood, particularly if that person is emotionally vulnerable. Any LLM or AI is trained on a dataset, and they are trained according to their overall accuracy on results; such training does not consider the needs of an individual. A human counselor or psychologist studies and trains for years to be able to detect and diagnose the ailment and correct treatment or response that a patient will need. How could we be sure an LLM will give the response needed by the individual and not merely offer a recommendation or platitude that it decides is statistically the best, based on a superficial attribute of the person such as their age, gender, or race?

Mitigating AI biases is not a simple task, but that does not mean we should not aspire to address the issue and work toward a better future. The responsibility to ensure duty of care falls to each person who is involved in creating, deploying, and using a system that can be used to treat or influence a patient—including the developer, the product owners, and the institution or medical professional that employs any device that uses such technologies. While the use of AI techniques such as LLMs in medical and healthcare products is relatively new, there are a number of standards and guidelines that these stakeholders must familiarize themselves with. The EU has recently reached a political agreement on the EU AI Act, to be finalized this year, which considers generative AI including LLMs. This was developed to ensure the regulated use of AI within the EU. NIST has also published a standard on identifying and managing bias in AI [65], and the IEEE plans to release the P7003 standard on Algorithmic Bias Considerations later this year [66], along with a certification course available through the IEEE CertifAIEd [67]. Ideally, there would be a simple coding fix to this problem; we could merely install a plug-in, a function or an update that would ensure all software would act in a fair and non-discriminatory manner. Unfortunately, no such technological fix exists, nor is it likely to ever exist. There are now a number of standards, guidelines, and regulations that developers must consider in order to ensure ethical and fair code generation, and it is up to each developer and user to make themselves aware of these guidelines and follow them to the best of their ability. If an LLM is employed as a chatbot that acts as a medical first-aider, or mental-health support bot, for instance, there is a duty to ensure it only offers fair, unbiased opinions or suggestions regardless of any sensitive or protected attribute from the user. LLMs are trained on real-world data, and the real world is biased. This is a simple fact that we must not just accept, but must work to overcome. Technology is here, and it is at our fingertips, but it is up to us to use it in a manner that helps all; we can only achieve this by considering the ethical and fairness implications of the use of these technologies on all people, at each step of development and deployment.

4.7 | Ethical Considerations in LLMs for Software Development

Other than bias, there are also some ethical concerns, including privacy breaches associated with LLMs that need to be considered. The ethical implications of LLMs discussed by experts are as follows:

Expert 1: At established scales such as enterprize customers, these models will have to deal with a lot of proprietary data bound by many data protection laws, such as the General Data Protection Regulation (GDPR) in the European Union (EU). For best ethical practices, LLM service providers would need to comply with such laws, for example, through on-premises clusters/instances for those customers to ensure data locality and protection. Further, they would need to come to an agreement with customers about the kind of data that can be used to train their models, both bespoke and general training.

Expert 5: The extensive use of LLMs by students can limit their ability to learn, think critically, and experiment with coding, and can raise ethical issues concerning cheating and plagiarism. The capacity of LLMs to enable plagiarism compromises academic integrity and undermines the goal of assessment, which is to impartially assess students' learning. Students who do excellent work using LLM models such as ChatGPT unfairly benefit from an advantage over their colleagues who do not have access to it. More importantly, using these models

interferes with teachers' ability to effectively assess student performance, making it challenging to address students' learning issues [68]. One other aspect of biases in LLMs is the language dominancy that hinders the explanation of the problem for non-native speakers, thus producing faulty outputs. Moreover, it also imposes some other ethical concerns, including dominance, breaching the privacy and confidentiality of individuals, dissemination of inaccurate information, and plagiarism. Italy was the first European country to ban ChatGPT by questioning the regulations for the storage and collection of users' data. Authorities also expressed disapproval for the improper handling of erroneous information generated by the platform [69]. Although the ban has been lifted now, it raises some serious concerns about the LLM models that are rigorously trained on text data without undergoing any preprocessing to preserve privacy, ensure transparency, mitigate biases, and handle inaccurate information that, in code generation, can limit assessability of functional correctness of the code generated [70]. It becomes crucial to strike a balance between minimizing ethical hazards and utilizing LLMs' powers in sensitive contexts. This calls for strict regulations and a close examination of how they are used.

4.8 | Challenges of Using LLMs in Software Development

Although LLMs have huge potential, there are also some challenges that need to be addressed as follows:

Expert 3: There is a lot of hype around AI at the moment, largely due to the attention economy that is capitalizing on recent advancements like PaLM 2, ChatGPT, Midjourney, and so on. Hype is not the same as recognizing the advancements in technology. Instead, the hype inflates expectations and confuses the real-world applications of AI with science fiction [71]. This can impact a population of workers in a number of ways. It makes them emotional, scared, fatigued, and finally skeptical (when hyped-up predictions fail to come to pass). Many AI experts are skeptical of contemporary AI replacing software developers, highlighting how the most advanced AI systems today, LLMs, are just statistical models. These don't have intelligence or understanding. Even if all software developers and companies used AI during development, the developer would still be required to design, review, and analyze the entire system. Rather than replacement, it is believed that it will be a tool to augment their work and abilities, which is known as intelligence augmentation [72]. As the technology improves, more tasks and responsibilities of a software developer may be automated by AI. But we don't know if the capabilities of AI will be so advanced that they can replace humans entirely. A possible future where AI is advanced enough to replace a human at tasks will not be the same as the world is now. We can't think about the future in terms of how the present is. In the same way that when the combustion engine was first invented, no one could have predicted how this would impact the world for the next century. While there were some paved roads, there was no expanse of highways. Likewise, the secondary industries that exist due to cars include motorsports, petrol stations, mechanics, and aftermarkets for spare parts and accessories. No one could have predicted the vastness of the car industry at the time of its birth, and AI will be the same. There will be other opportunities that come with advanced AI. We have the benefit of learning from the past. Software developers should be aware that we are now in a revolutionary time, where the world is slowly starting to shift into a new trajectory.

Expert 7: LLMs serve as a driving force behind the transformative change in developers' methodologies for conceptualizing, crafting, and enhancing code in software development endeavors. Similar to how automation rendered much of manual testing obsolete, the strategic utilization of LLMs holds promise for augmenting productivity and elevating the quality standards of software products. However, despite the proliferation of available options, many LLMs remain nascent in their capabilities, thus constraining their current applicability. Moreover, Instruction-tuned LLMs [73] like OpenAI's ChatGPT emerged as an early frontrunner in the conversational AI market, establishing widespread adoption until the introduction of Google's Bard, a successor of pathways language model 2 (PaLM 2) [74]. Utilizing both platforms extensively, developers tend to encounter a sense of disillusionment with ChatGPT's performance. Particularly within the context of interfacing with Windows services and applications, expectations were set for more dependable responses and proficient generation of code snippets tailored for Microsoft-owned libraries. While challenges without immediate resolutions are inherent in software development, the provision of inaccurate or irrelevant solutions is considered unacceptable. While prompting ChatGPT to suggest or create code snippets for web application development within the Microsoft Blazor framework [75], it was found that the generated code block ostensibly addresses specific frontend properties that were absent in the framework. Furthermore, comparative evaluations between ChatGPT and Bard have highlighted instances where the latter has offered more appropriate solutions, intensifying dissatisfaction with the former's performance.

Expert 8: It is found that there is a lack of research on the use of LLMs in software engineering, especially in areas like software testing, turning natural language into code, and incorporating newer models like ChatGPT. Integrating LLMs into software engineering is a complicated process. This stresses the importance of conducting thorough literature reviews to understand the capabilities and limitations of LLMs which can bridge this knowledge gap [76]. Despite the advantages, there are also several challenges and limitations when it comes to generating code using language models. The LLMs struggle with retaining the testing context, which makes it difficult to generate accurate and relevant test scripts within the dynamic mobile app testing environment. LLMs may use inappropriate or deprecated APIs, which can lead to non-executable scripts and necessitate manual intervention to align scripts with current APIs. It is also found that LLMs require human expertise for fine-tuning scripts to ensure compatibility with the mobile app's environment, due to a lack of deep understanding of app intricacies. LLMs have limited capabilities in generating scripts for complex test events like scrolling or managing pop-ups, requiring additional tools or manual efforts for comprehensive test coverage [77].

Expert 10: LLM augmented development already looks set to accelerate rapidly in 2024, with new LLM models and tools being announced almost weekly. Alongside the now well-established tools such as Copilot, Code Whisperer, and Gemini, other tools are regularly coming to market with new capabilities. However, some of the concerns about the proprietary models used to power these tools are that they are all commercial, opaque, and require internet access. Additionally, they still have unresolved questions about copyright. A counter to these concerns is the growth of open-source models such as Llama, Mistral, and Dolphin, which are opening up LLM development to far more participants in both industry and academia. This will allow for far more research and innovation in the specialist area of AI coding tools. While current coding assistants have proven very effective in code production, they are also still susceptible to excessive and irrelevant code as well as hallucinations. They are also still limited in terms of software design [78].

4.9 | Future Directions and Innovations in LLMs for Software Development

LLMs have also opened up various future opportunities for innovation to leverage the true potential of these architectures. Some of the opportunities from expert opinions are listed below:

Expert 1: Currently, the biggest blocker is scaling these models. For an incremental improvement in the dataset size or efficiency, it requires a lot of computing power. This makes these extremely useful technologies a difficult sell at an enterprize level. Now that we're approaching a saturation point in Moore's law and the number of transistors we can fit in conventional silicon, we need to invest in silicon chips that follow the principles of a human brain. It will help democratize this breakthrough technology for the masses. The pricing models for such LLMs are based on the number of tokens used per prompt. As Prompt engineering is becoming an up-and-coming job profile, we can train engineers to be more efficient in their prompts and extract maximum value out of prompts. Currently, the feedback loop of these models for media-based inputs (such as pictures, videos, or audio) is long. With investment toward human-brain-like chips, we can reduce this feedback to reach near real-time response from LLMs for a wide array of input types.

Expert 6: Despite the current productivity gains and performance of code language models, their full potential remains unrealized. Due to their unreliability, current models are primarily utilized for automating repetitive tasks and providing skeletal structures for ideas, simplifying the ideation process. However, a significant transformation in the software development pipeline is still on the horizon. Ethical considerations come into play as these code models could inadvertently aid in the development of malicious software, amplifying the ease of scaling such efforts. Future development efforts hinge on designing more reliable models. Key considerations include obtaining high-quality code data and fine-tuning language models to adhere to specific software development patterns and best practices, thereby minimizing errors.

Expert 10: The speed of evolution of new LLMs makes it difficult to predict exactly what technologies will be powering 3GL code generation tools, even in the near term. But what is easier to predict is the drivers of this technology, which remain relatively unchanged. Businesses will always aspire to convert business ideas to IT solutions with the lowest possible cost and time to market. The popularity of Low Code was on the back of this promise, but to date has not been able to deliver. AI tools may finally make it possible to convert business ideas to solutions with little or no need for traditional development. This could precipitate a realignment of solution delivery roles. The role of developers who currently convert business requirements to code could move to no-code solutions, developed by "business engineers" who are primarily aligned to business functions rather than IT. These would be technical experts fully embedded in the business. They may still require some traditional coding skills (for instance, Excel has just embedded Python), but not in the way they do as application developers. Data scientists may increasingly occupy this space as many AI tools automating functions that data scientists currently need to code are coming to market. Operations has evolved significantly over the last decade with DevOps, but there are already signs that this will evolve further and realign into Development and Platform Engineering. Platform engineering is responsible for developing tooling to run infrastructure, production, and the development environment, abstracting away the complexity of operations from developers. This is also a further enabler of moving solution development to business functions. At the same time, Platform engineers will require increasingly advanced AI coding tools. These will need all the development features of existing AI code, but in addition will need tools for IaC, configuration, security, testing, production management, and other pipeline toolsets. AI Ops skills may likely need this level of expertise to manage and deploy models. Finally, AI model developers and AI tool developers will require a range of existing and new AI tools to facilitate training, testing, and management of new models. Already, these are using LLMs to generate new LLMs, and if this proves productive, no doubt new developer tools will emerge to accelerate these processes. It is possible that the future of AI coding tools may be shaped as much by the future landscape of development roles in the AI age as it is by the evolution of the AI tools we have today.

Expert 11: LLMs have numerous benefits, but they cannot completely replace human interaction. As a result, it is recommended that LLM models be used in conjunction with the manual CR process, with LLM handling mundane and repetitive tasks and human developers focusing on higher-order, creative problem-solving.

5 | Overview, Analysis, and Discussion

This section provides a brief overview of the major themes discussed by experts.

5.1 | Thematic Analysis

This section presents a thematic analysis of the expert opinions collected in the study. The analysis is based on themes

TABLE 3 | Major themes discussed by the experts.

Theme	1	2	3	4	5	6	7	8	9	10	11
Compliance		✓					✓				
Ethics and fairness	✓			✓	✓					✓	
Lack of innovation and creativity					✓						
Human–AI collaboration			✓			✓					
Increase productivity									✓		
Code review	✓	✓				✓					
Code quality				✓		✓		✓	✓		
NLG and NLU	✓										
Lack of contextual understanding				✓					✓	✓	
Collaboration and inclusivity	✓			✓							✓
Security concerns	✓				✓						
Non-compliance				✓					✓		

TABLE 4 | Cohen's κ for inter-annotator agreement between themes.

Theme	κ score
Code quality	0.2142
Code review	0.7441
Collaboration and inclusivity	0.7441
Compliance	1.0
Ethics and fairness	0.3773
Human–AI collaboration	0.6206
Improve productivity	0.2978
Lack of contextual understanding	−0.1578
Lack of innovation and creativity	1.0
NLG and NLU	−0.1379
Non-compliance with industry standards	1.0
Security concerns	0.6206
Mean	0.5269

identified by two annotators through iterative reading, discussion, and subsequent agreement between the two annotators. Rather than following a formal qualitative thematic coding methodology, the annotators collaboratively reviewed the responses, identified prominent themes, and reached consensus on the final themes that captured common viewpoints, perceived advantages, eminent concerns, and areas of emphasis across experts. This approach aligns with the exploratory and opinion-oriented nature of the study rather than in-depth qualitative exploration.

The analysis revealed a set of recurring themes that reflect how experts perceive the role of LLMs in software development. These themes span technical, organizational, administrative, and ethical dimensions, highlighting both opportunities and challenges associated with LLM adoption in the development workflow. In total, 12 common themes have been identified to perform thematic analysis, which are as follows: Compliance, Ethics and Fairness, Lack of Innovation and Creativity, Human–AI Collaboration, Increase in Productivity, Code Review, Code Quality, NLG and NLU, Lack of Contextual Understanding, Collaboration and Inclusivity, Security Concerns, and Non-Compliance to Industry Standards. Table 3 displays the corresponding themes for each expert. The most common themes are "Ethics and Fairness" and "Code Quality," as out of 11 experts, 4 talk about the potential ethical considerations and fairness in the application of LLMs in software development and their impact on code quality. Only one expert explicitly discusses the "NLG and NLU" capabilities of LLMs.

Table 4 displays the kappa coefficients for the 12 thematic codes annotated by two annotators. The κ score ranges from −1 to +1, with +1 conveying perfect agreement and −1 indicating less than chance agreement. "Compliance," "Lack of innovation and creativity," and "Non-compliance to industry standards" have perfect agreement between the two annotators, whereas "Lack of contextual understanding" and 'NLG and NLU' have less than chance agreement. The mean kappa score is 0.5269, which indicates moderate agreement between the annotator assigned labels. While not all experts discussed every theme, several topics appeared consistently across responses, indicating shared concerns and expectations.

1. *Compliance and non-compliance to industry standards*: Compliance with industry standards and best practices emerged as an important consideration in the adoption of LLMs for software development amongst the opinion essays. Experts noted that while LLMs can assist in enforcing coding standards and formatting guidelines, they do not inherently guarantee compliance with domain-specific regulations or organizational policies, and may sometimes be detrimental to that goal without human supervision.

2. *Ethics and fairness*: Ethics and compliance emerged as significant cross-cutting themes. Experts expressed concerns about bias embedded in training data and the potential for LLMs to propagate unfair or discriminatory outcomes. Compliance with industry standards and responsible AI principles was also highlighted, particularly in regulated environments. Some experts raised concerns about noncompliance, especially when LLM-generated code is used without sufficient review or documentation

3. *Lack of innovation and creativity*: A smaller but notable theme relates to the potential impact of LLMs on the creativity of developers and users. While some experts viewed LLMs as enablers of rapid prototyping, brainstorming, and productivity enhancers, others cautioned that overreliance on automated suggestions could lead to homogenized solutions, reduced creativity and innovation in software design.

4. *Human–AI collaboration and inclusivity*: Another prominent theme extracted from the essays was concerns about the collaborative relationship between developers and LLMs. Experts consistently emphasized that LLMs should be used as assistive tools rather than replacements for human developers. Effective human–AI collaboration was seen as the essential step to realise the benefits of LLMs while keeping the risks to a minimum. Several experts emphasized that developers must retain the final decision-making authority, particularly in critical choices and scenarios. The essays also emphasized how LLMs within the development environment can enhance collaboration between experts, as well as promote inclusivity by encouraging participants from experts across the world in cross-lingual scenarios.

5. *Increase in productivity*: One of the most frequently identified themes was 'Increased Productivity'. Experts emphasized the ability of LLMs to automate time-consuming tasks such as code generation, documentation, debugging, and evaluation. These capabilities were seen as particularly valuable in reducing development time and effort, allowing developers to focus on higher-level problem-solving and reducing cognitive load. However, productivity gains were often discussed alongside the need for careful oversight and the need for human-in-the-loop development

6. *Code review and code quality*: Experts widely acknowledged the effectiveness of LLMs on streamlining the CR processes and enhancing overall code quality. LLMs were viewed as useful tools for identifying syntactic errors, debugging, and suggesting improvements. Some experts noted that automated and instant support can enhance consistency and reduce human error, but concerns were also raised about the limitations of LLMs in understanding complex problems, reinforcing the need for human oversight.

7. *NLG and NLU*: Experts frequently discussed the advantages of the enhanced NLU and NLG capabilities of LLMs within software development. LLMs are particularly effective at translating natural language requirements into code snippets and generating documentation. This capability was seen as valuable for bridging communication gaps between technical and nontechnical stakeholders, enabling improved collaboration

8. *Lack of contextual understanding*: Several experts noted that LLMs often lack deep contextual understanding of domain-specific requirements or organizational constraints. They also lose track of context within a long conversation, which can be detrimental to code development and may introduce bugs into the code.

9. *Security concerns*: Security-related concerns were identified by multiple experts, including the risk of introducing vulnerabilities through AI-generated code. Experts expressed concern that developers may place unwarranted trust in AI-generated code and potentially ignore standard security reviews or testing procedures. This risk is amplified when LLMs are used to rapidly generate large volumes of code to be used in critical scenarios. As a result, experts emphasized the need to integrate LLM usage with standard and secure development practices, involving CRs, static analysis, and security testing.

5.2 | Opinions Overview

The role of LLMs in automating or assisting code generation is significant. Our expert panel has highlighted several advantages and use cases where LLMs can make a huge difference in software development. Four experts acknowledged that integrating these models in code generation activities can reduce the chances of errors in both syntax and logic, thus improving the code quality. Also, Experts 1 and 4 have mentioned that these platforms can aid cross-language collaboration, that is, code written in one programming language can be translated into other languages easily, which increases usability and reduces effort. Moreover, Experts 4 and 8 mention that LLMs can generate code snippets based on statements written in plain language or natural language, increasing the LLMs' applicability span. In the view of Expert 1, companies like Amazon, Microsoft, Google, and Facebook (Meta) are big players in developing these platforms to enhance developer productivity and experience. Furthermore, Expert 8 draws attention to the ability of LLMs to create problem-specific software development or code generation that helps developers in efficient time utilization and reduces their burden. Figure 1 shows the names of the most common LLMs discussed by the expert panel. GPT is the most frequent LLM mentioned, followed by Co-Pilot and Llama.

LLMs have greatly impacted the CR process. According to Expert 11, the incorporation of LLMs can enhance communication and collaboration between developers. An automated CR process can increase the effectiveness and accuracy of the code as well as the CR process. It can help to ensure best practices, thus improving the overall code quality as recognized by Experts 1 and 2. Moreover, Expert 2, being a practitioner in the medical industry, says that the LLMs' CR process can reduce the risk by mitigating defects in the early stage of medical device software development. Another important aspect that was brought forward by Expert 2 is related to security, as these models can also check codes for vulnerability detection and security audits. It can also help prioritize and optimize resource allocation.

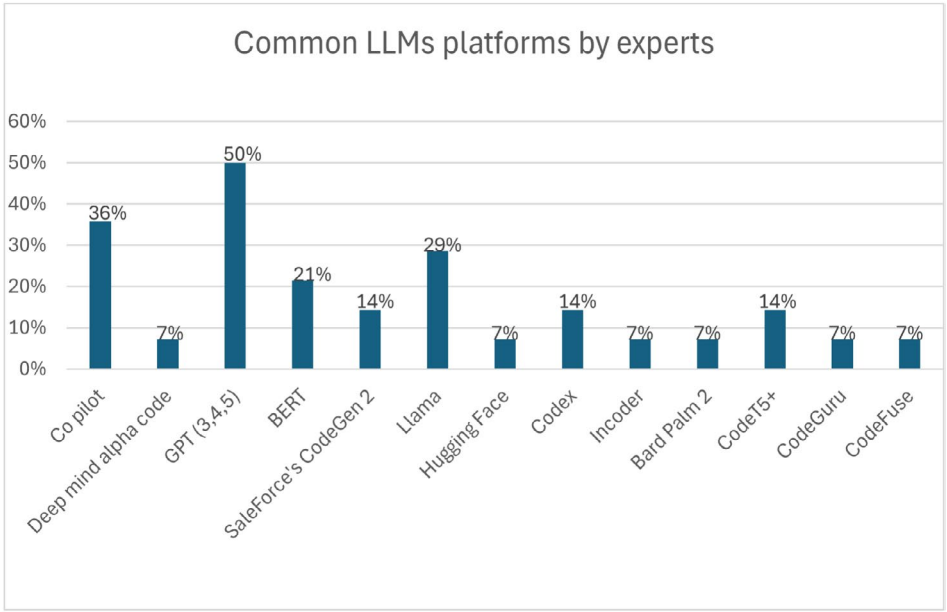

FIGURE 1 | Most common LLMs discussed by the expert panel.

Expert 1 also draws a view on the predictive analysis of LLMs. The data-driven predictions made by these language models can empower software development processes, help in optimized resource allocation, and provide assistance in long-term innovation and improvement in the software community.

One of the major concerns highlighted by experts is ethical and fairness issues, as these models are trained on a large corpus of internet data. The data can be biased, and these biases can be reflected in the form of gender stereotypes, religion, or social prejudice. The impact of these biases can highly affect a specific group when implemented in systems such as hiring software or medical care applications, as mentioned by Experts 5 and 8. Other than that, the applications of open-source LLMs have raised ethical concerns for their extensive use by students. It limits the learning and innovation process. Furthermore, Experts 1 and 5 also highlight that these models can breach privacy and data protection laws as they can memorize new data. Experts 5 summarize that these models lack transparency and accountability. Also, the underlying bias can lead to discriminatory outputs.

There are several other limitations mentioned by the experts. These mainly include the inability of language models to handle ambiguity and vague instructions, which makes it difficult to generate correct, accurate, and relevant scripts. Also, LLMs are trained on a large amount of text on the internet; therefore, the code generated by them lacks specific industry standards or regulations. Also, these generated code snippets may not align with new APIs. Experts 3 and 11 express reservations about the applicability of these models, noting that viewing them as a replacement for human beings can bring anxiety and depression to the workforce. It is important to acknowledge that software development still requires humans to perform tasks such as higher-order planning, code verification, and fine-tuning of the code.

The full potential of LLMs is still not realized. Expert 1 says that these models should be trained on large datasets, and also that developers should be trained to use prompts efficiently to get the maximum from them. Expert 6 believes that these models should be trained to increase the reliability of the content generated by them by increasing data exposure and testing protocols.

6 | Discussion

In this section, we will discuss the findings of the study with respect to the research questions:

1. RQ1: What are the applications and advantages of LLMs in software development as per expert opinions? The thematic analysis approach applied in this study has revealed several perceived benefits that the integration of LLMs in software development can provide. LLMs have been used to perform several tasks in software development, such as code reviewing and pair programming [79], code generation, converting from natural language prompts to code, generating documentation for the code, improving code quality [76], translating code across programming languages, and predicting code snippets in real time. LLM-assisted platforms such as GitHub Copilot can serve as valuable assistants for developers by automating tedious tasks and allowing them to focus on tasks that require creative problem solving (Eirini [27]). Within a predefined framework, LLMs can help ensure compliance with standards [76]. They can also help in maintaining version control for the various dependencies in an application. LLMs can also support cross-platform and cross-lingual collaboration between developers.
2. RQ2: What are the limitations and potential concerns of LLMs in software development as per expert opinions? Despite seeing extensive application in software

development, there are still significant drawbacks to using LLMs, as affirmed by the expert opinions. Firstly, LLMs suffer from "hallucinations," which refers to generating factually incorrect information [80]. Therefore, responses provided by LLMs should always be verified. Secondly, overreliance on LLMs can limit the critical thinking skills of new developers and hamper innovation. Students can also unfairly use LLMs to complete assignments or tests [39]. Vague instructions given to LLMs may result in incorrect scripts. There are also significant ethical considerations to the application of LLMs in software development [36]. Deep learning algorithms contain inherent biases which may be amplified with downstream applications. LLMs are pretrained on a large amount of data, which also contains biases such as gender bias [81], racial bias, and brilliance bias. Integration of LLMs into software development workflows may also disrupt version control and compliance, model transparency, and responsible AI practices. Additional training may also be required to efficiently use the LLMs.

6.1 | Implications of the Findings

The findings suggest that LLMs have the potential to meaningfully reshape software development practices, but only when integrated ethically and with due consideration. The results highlight the importance of balancing automation with accountability and human oversight. LLMs can accelerate development workflows, but also introduce new risks which can impact trust, explainability, and ethical responsibility. For research, the study underscores the value of expert opinions as an early indicator of emerging technologies such as AI and their associated hype and concerns. The identified themes point to the need for further empirical and qualitative research on how LLM usage affects code quality, developer expertise, training and education, and organizational decision-making.

6.2 | Practical Recommendations for Practitioners and Researchers

Based on the synthesized expert perspectives, several practical recommendations emerge for software development practitioners and researchers: First, LLMs should be used as assistive tools under the developer's supervision, rather than autonomous decision-makers. Human review and validation remain essential. Second, organizations should invest in developer training and education, emphasizing critical evaluation of AI-generated outputs, rather than blindly accepting LLM-generated code. Third, ethical considerations, including bias awareness, data provenance, privacy and security, and responsible AI practices, should be embedded into software development workflows incorporating LLMs. Finally, administrators and policymakers should establish clear guidelines on the appropriate use of LLMs in educational and professional settings to prevent misuse, plagiarism, and overdependence.

Based on the analysis of the expert opinions performed in the study, LLMs have the ability to efficiently perform tasks in the software development lifecycle. However, their integration into existing workflows must be done while keeping the various ethical, security, and privacy concerns in mind. The study analyzed opinions belonging to experts from a varied demographic, and from the industry and academia sectors. However, additional work must be conducted by considering different populations to ensure that the findings of the study generalize to other groups. Exploring additional tools, such as in-person interviews and structured surveys, may allow an in-depth analysis of the opinions of experts in the field of software development and NLP.

There are several limitations to this study. The opinions were gathered in 2023, when the use of LLMs in software development was still relatively niche. Therefore, this paper lacks the expert opinions on newer models and tools which we will address in our future work. As of 2026, we have followed up with our experts regarding the use of agentic AI in software development activities. Most respondents continue to prefer LLMs for code generation over agentic AI systems due to reasons such as debugging complexity, predictability, security risks, and reduced control over system behavior.

7 | Conclusion and Future Work

Since the launch of LLMs, they have been utilized for various applications such as image generation, healthcare, education, and software development. LLMs such as CodeBERT and Codex can be used for various tasks in the software development lifecycle such as CR, generation of documentation, pair programming, and so on. Despite the numerous advantages, there are several limitations as well as ethical implications to their use, and any integration with existing frameworks needs to be carefully monitored. This paper presented an opinion-driven synthesis and cursory thematic analysis of expert perspectives on the influence of LLMs in software development. By consolidating insights from practitioners and researchers, the study identified both the opportunities and challenges associated with LLM-assisted development, including productivity gains, enhanced collaboration, improvements in code quality, ethical and fairness risks, and concerns related to overreliance and developer satisfaction. The primary contribution of this work to the field of applied AI lies in its analysis of first-hand expert perspective on a rapidly evolving technological landscape. Rather than focusing solely on algorithmic performance and quantitative evaluation, the paper highlights practical and human-centric considerations critical for the responsible and ethical application of LLMs in real-world software engineering scenarios. The analysis of expert opinions can be a good indicator of emerging practices, risks, and research directions in applied AI. Moving forward, it is important to understand that LLMs are based on algorithms that contain bias, which can be further propagated through irresponsible use. Future work can extend this research by conducting deeper thematic analyses using structured qualitative methods and thematic coding, expanding the diversity and size of the expert participant pool, and examining the long-term impacts of LLM integration on software development practices through quantitative studies. Also, we will extend the scope to the use of agentic AI in software development and the latest advancements in LLMs.

Acknowledgments

The project is partially supported by the DkIT Postgraduate Scholarship and Taighde Éireann—Research Ireland under Grant number 13/RC/2094_2 and Grant number 21/FFP-A/9255.

Funding

This work was supported by DKIT and Taighde Éireann—Research Ireland, 13/RC/2094_2, 21/FFP-A/9255.

Ethics Statement

This manuscript presents an expert-opinion-based analysis of the influence of large language models (LLMs) in software development. The study was conducted according to responsible research and publication practices. Expert contributions were analyzed solely for research purposes. The authors acknowledge ethical considerations surrounding LLM adoption, including bias, reliability, accountability, and responsible deployment, and these issues are addressed through balanced analysis and transparent reporting.

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

1. T. Brown, B. Mann, N. Ryder, et al., "Language Models Are Few-Shot Learners," *Advances in Neural Information Processing Systems* 33 (2020): 1877–1901.

2. C. Dong, Y. Li, H. Gong, et al., "A Survey of Natural Language Generation," *ACM Computing Surveys* 55, no. 8 (2022): 1–38.

3. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding," *arXiv* 1 (2018): 4171–4186.

4. A. Radford, K. Narasimhan, T. Salimans, et al., "Improving Language Understanding by Generative Pre-Training," OpenAI, (2018).

5. R. Luo, L. Sun, Y. Xia, et al., "Biogpt: Generative Pre-Trained Transformer for Biomedical Text Generation and Mining," *Briefings in Bioinformatics* 23, no. 6 (2022): bbac409.

6. Z. Liu, F. Jiang, Y. Hu, C. Shi, and P. Fung, "Ner-Bert: A Pre-Trained Model for Low-Resource Entity Tagging," *arXiv* (2021).

7. S. Clinchant, K. W. Jung, and V. Nikoulina, "On the Use of Bert for Neural Machine Translation," *arXiv* (2019): 108–117.

8. G. Kaur, A. Kaushik, and S. Sharma, "Cooking Is Creating Emotion: A Study on Hinglish Sentiments of Youtube Cookery Channels Using Semi-Supervised Approach," *Big Data and Cognitive Computing* 3, no. 3 (2019): 37.

9. S. R. Shah, A. Kaushik, S. Sharma, and J. Shah, "Opinion-Mining on Marglish and Devanagari Comments of Youtube Cookery Channels Using Parametric and Non-Parametric Learning Models," *Big Data and Cognitive Computing* 4, no. 1 (2020): 3.

10. S. Yadav and A. Kaushik, "Comparative Study of Pre-Trained Language Models for Text Classification in Smart Agriculture Domain," in *Advances in Data-Driven Computing and Intelligent Systems: Selected Papers From ADCIS 2022*, vol. 2, ed. S. Das, S. Saha, C. A. Coello Coello, and J. C. Bansal, (Springer, 2023), 267–279.

11. S. Yadav, A. Kaushik, M. Sharma, and S. Sharma, "Disruptive Technologies in Smart Farming: An Expanded View With Sentiment Analysis," *AgriEngineering* 4, no. 2 (2022): 424–460.

12. A. Hörnemalm, "Chatgpt as a Software Development Tool: The Future of Development," Master's Thesis, Umea University (2023).

13. S. S. Biswas, "Role of Chat GPT in Public Health," *Annals of Biomedical Engineering* 51, no. 5 (2023): 868–869.

14. I. Adeshola and A. P. Adepoju, "The Opportunities and Challenges of Chatgpt in Education," *Interactive Learning Environments* 32 (2023): 1–14.

15. A. Kaushik, S. Yadav, A. Browne, et al., "Exploring the Impact of Generative Artificial Intelligence in Education: A Thematic Analysis," *arXiv* (2025).

16. A. Kaushik, S. Yadav, S. Sharma, and K. McDaid, "Harvesting Insights: Sentiment Analysis on Smart Farming Youtube Comments for User Engagement and Agricultural Innovation," *IEEE Technology and Society Magazine* 43, no. 3 (2024): 91–100, https://doi.org/10.1109/MTS.2024.3455754.

17. R. R. Kovvuri, A. Kaushik, and S. Yadav, "Disruptive Technologies for Smart Farming in Developing Countries: Tomato Leaf Disease Recognition Systems Based on Machine Learning," *Electronic Journal of Information Systems in Developing Countries* 89, no. 6 (2023): e12276.

18. S. Yadav and A. Kaushik, "Comparative Study of Pre-Trained Language Models for Text Classification in Smart Agriculture Domain," in *Advances in Data-Driven Computing and Intelligent Systems*, ed. S. Das, S. Saha, C. A. Coello Coello, and J. C. Bansal (Springer Nature, 2023), 267–279.

19. S. Yadav, A. Kaushik, S. Sharma, and K. McDaid, "The Future of Agriculture: Analysing User Sentiment on Smart Farming With Explainable Artificial Intelligence," in *IEEE International Symposium on Technology and Society (ISTAS)*, ed. F. Caraffini (IEEE, 2023), 1–8, https://doi.org/10.1109/ISTAS57930.2023.10306134.

20. J. Achiam, S. Adler, S. Agarwal, et al., "GPT-4 Technical Report," *arXiv* (2023).

21. M. Chen, J. Tworek, H. Jun, et al., "Evaluating Large Language Models Trained on Code," *arXiv* (2021).

22. Z. Feng, D. Guo, D. Tang, et al., "Codebert: A Pre-Trained Model for Programming and Natural Languages," *arXiv* (2020): 1536–1547.

23. V. Raychev, M. Vechev, and E. Yahav, "Code Completion With Statistical Language Models," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ed. K. Pingali (ACM, 2014), 419–428.

24. A. Desai, S. Gulwani, V. Hingorani, et al., "Program Synthesis Using Natural Language," in *Proceedings of the 38th International Conference on Software Engineering*, ed. W. Visser and L. Williams (ACM, 2016), 345–356.

25. Z. Zhao, Z. Xu, J. Zhu, P. Di, Y. Yao, and X. Ma, "The Right Prompts for the Job: Repair Code-Review Defects With Large Language Model," *arXiv* (2023).

26. X. Pu, M. Gao, and X. Wan, "Summarization Is (Almost) Dead," *arXiv* (2023).

27. E. Kalliamvakou, "Research: Quantifying GitHub Copilot's Impact on Developer Productivity and Happiness," 2022, https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/.

28. Z. Zhang, M. Rayhan, T. Herda, M. Goisauf, and P. Abrahamsson, "LLM-Based Agents for Automating the Enhancement of User Story Quality: An Early Report," *arXiv* (2024): 117–126.

29. M. Liffiton, B. E. Sheese, J. Savelka, and P. Denny, "Codehelp: Using Large Language Models With Guardrails for Scalable Support in Programming Classes," in *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, ed. I. Jormanainen (ACM, 2023), 1–11.

30. W. Lyu, Y. Wang, T. Chung, Y. Sun, and Y. Zhang, "Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study," in *Proceedings of the Eleventh ACM Conference on Learning@Scale*, ed. M. K. Kim, X. Wang, and M. Xia (ACM, 2024), 63–74.

31. D. B. Acharya, K. Kuppan, and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey," *IEEE Access* 13 (2025): 18912–18936.

32. L. Gao, S. Biderman, S. Black, et al., "The Pile: An 800GB Dataset of Diverse Text for Language Modeling," *arXiv* (2020).

33. G. Deng, Y. Liu, V. Mayoral-Vilches, et al., "PentestGPt: An LLM-Empowered Automatic Penetration Testing Tool," *arXiv* (2023).

34. D. Huang, Q. Bu, J. Zhang, X. Xie, J. Chen, and H. Cui, "Bias Assessment and Mitigation in LLM-Based Code Generation," *arXiv* (2023).

35. Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A Survey on Large Language Model (LLM) Security and Privacy: The Good, the Bad, and the Ugly," *High-Confidence Computing* 4 (2024): 100211.

36. U. P. Liyanage and N. D. Ranaweera, "Ethical Considerations and Potential Risks in the Deployment of Large Language Models in Diverse Societal Contexts," *Journal of Computational Social Dynamics* 8, no. 11 (2023): 15–25.

37. L. Tjuatja, V. Chen, S. Tongshuang Wu, A. Talwalkar, and G. Neubig, "Do LLMs Exhibit Human-Like Response Biases? A Case Study in Survey Design," *arXiv* 12 (2023): 1011–1026.

38. M. A. Kuhail, S. S. Mathew, A. Khalil, J. Berengueres, and S. J. H. Shah, ""Will I Be Replaced?" Assessing Chatgpt's Effect on Software Development and Programmer Perceptions of AI Tools," *Science of Computer Programming* 235 (2024): 103111.

39. M. Perkins, "Academic Integrity Considerations of Ai Large Language Models in the Post-Pandemic Era: Chatgpt and Beyond," *Journal of University Teaching & Learning Practice* 20, no. 2 (2023): 7.

40. S. Pudasaini, L. Miralles-Pechuán, D. Lillis, and M. Llorens Salvador, "Survey on Ai-Generated Plagiarism Detection: The Impact of Large Language Models on Academic Integrity," *Journal of Academic Ethics* 23 (2024): 1–34.

41. V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe, "Software Engineering Education Must Adapt and Evolve for an LLM Environment," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education*, ed. L. Battestilli, S. A. Rebelsky, and L. Shoop (ACM, 2024), 666–672.

42. A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, "On the Naturalness of Software," *Communications of the ACM* 59, no. 5 (2016): 122–131.

43. I. Ozkaya, "Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications," *IEEE Software* 40, no. 3 (2023): 4–8.

44. H. Koziolek and A. Koziolek, "LLM-Based Control Code Generation Using Image Recognition," *arXiv* (2023): 38–45.

45. J. Li, G. Li, Y. Li, and Z. Jin, "Structured Chain-of-Thought Prompting for Code Generation," *arXiv* 34, no. 2 (2023): 1–23.

46. A. Q. Jiang, A. Sablayrolles, A. Roux, et al., "Mixtral of Experts," *arXiv* (2024).

47. Builder.io, "How Builder Works: A Technical Overview," accessed March 22, 2024, https://www.builder.io/c/docs/how-builder-works-technical#how-builder-works-a-technical-overview.

48. Y. Liu, J. Chen, T. Bi, et al., "An Empirical Study on Low Code Programming Using Traditional vs Large Language Model Support," *arXiv* 234 (2024): 112727.

49. R. Taori, I. Gulrajani, T. Zhang, et al., "Alpaca: A Strong, Replicable Instruction-Following Model," *Stanford Center for Research on Foundation Models* 3, no. 6 (2023): 7.

50. J. Sametinger, J. Rozenblit, R. Lysecky, and P. Ott, "Security Challenges for Medical Devices," *Communications of the ACM* 58, no. 4 (2015): 74–82.

51. H. K. Dam, T. Tran, and T. Pham, "A Deep Language Model for Software Code," *arXiv* (2016).

52. M. Madera and R. Tomoń, "A Case Study on Machine Learning Model for Code Review Expert System in Software Engineering," in *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, ed. M. Ganzha, L. Maciaszek, and M. Paprzycki (IEEE, 2017), 1357–1363.

53. R. Lou, K. Zhang, and W. Yin, "Is Prompt All You Need? No. A Comprehensive and Broader View of Instruction Learning," *arXiv* (2023).

54. A. Johnson, "The Role of Large Language Models in Code Review," 2023, https://medium.com/andrew_johnson_4/the-role-of-large-language-models-in-code-review-2b74598249ab.

55. E. Ntoutsi, P. Fafalios, U. Gadiraju, et al., "Bias in Data-Driven Artificial Intelligence Systems—An Introductory Survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, no. 3 (2020): e1356.

56. N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel, "Large Language Models Struggle to Learn Long-Tail Knowledge," in *International Conference on Machine Learning (PMLR)*, ed. S. Sabato and J. Scarlett (MLR Press, 2023), 15696–15707.

57. J. Gesi, X. Shen, Y. Geng, Q. Chen, and I. Ahmed, "Leveraging Feature Bias for Scalable Misprediction Explanation of Machine Learning Models," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, ed. L. Pollock and M. Di Penta (IEEE, 2023).

58. J. Shihadeh, M. Ackerman, A. Troske, N. Lawson, and E. Gonzalez, "Brilliance Bias in GPT-3," in *2022 IEEE Global Humanitarian Technology Conference (GHTC)*, ed. E. M. Belding (IEEE, 2022), 62–69.

59. B. Lorentzen, "Social Biases in Language Models: Gender Stereotypes in GPT-3 Generated Stories," Uppsala Universitet, (2022).

60. Y. Zhang, "The Impact of ChatGPT on HR Recruitment," *Journal of Education, Humanities and Social Sciences* 19 (2023): 40–44.

61. K. Saito, A. Wachi, K. Wataoka, and Y. Akimoto, "Verbosity Bias in Preference Labeling by Large Language Models," *arXiv* (2023).

62. R. Dresser, "Wanted Single, White Male for Medical Research," *Hastings Center Report* 22, no. 1 (1992): 24–29.

63. D. S. Char, N. H. Shah, and D. Magnus, "Implementing Machine Learning in Health Care—Addressing Ethical Challenges," *New England Journal of Medicine* 378, no. 11 (2018): 981.

64. C. Blease and J. Torous, "ChatGPT and Mental Healthcare: Balancing Benefits With Risks of Harms," *BMJ Mental Health* 26, no. 1 (2023): e300884.

65. R. Schwartz, A. Vassilev, K. Greene, L. Perine, A. Burt, and P. Hall, *Towards a Standard for Identifying and Managing Bias in Artificial Intelligence* (National Institute of Standards and Technology, 2022).

66. A. Koene, L. Dowthwaite, and S. Seth, "IEEE P7003™ Standard for Algorithmic Bias Considerations: Work in Progress Paper," *Proceedings of the International Workshop on Software Fairness* (2018): 38–41.

67. IEEE CertifAIEd, "IEEE CertifAIEd Professional Certification," accessed March 22, 2024, https://engagestandards.ieee.org/ieeecertifaied.html.

68. D. R. E. Cotton, P. A. Cotton, and J. R. Shipway, "Chatting and Cheating: Ensuring Academic Integrity in the Era of ChatGPT," *Innovations in Education and Teaching International* 61, no. 2 (2023): 1–12.

69. Italy Ban, "Italy Lifts Ban on ChatGPT After Data Privacy Improvements," 2022, https://www.dw.com/en/ai-italy-lifts-ban-on-chatgpt-after-data-privacy-improvements/a-65469742.

70. J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is Your Code Generated by Chatgpt Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," *Advances in Neural Information Processing Systems* 36 (2023): 21558–21572.

71. E. Siegel, "The AI Hype Cycle Is Distracting Companies," *Harvard Business Review* 2 (2023): 128.

72. R. Biedron, "Intelligence Augmentation vs Artificial Intelligence: What's the Difference?," accessed March 22, 2024, https://planergy.com/blog/intelligence-augmentation-vs-artificial-intelligence/.

73. L. Ouyang, J. Wu, X. Jiang, et al., "Training Language Models to Follow Instructions With Human Feedback," *Advances in Neural Information Processing Systems* 35 (2022): 27730–27744.

74. R. Anil, A. M. Dai, O. Firat, et al., "PaLM 2 Technical Report," *arXiv* (2023).

75. Blazor, "Build Beautiful Web Apps With Blazor," accessed March 22, 2024, https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor.

76. X. Hou, Y. Zhao, Y. Liu, et al., "Large Language Models for Software Engineering: A Systematic Literature Review," *arXiv* 8 (2023): 1–79.

77. S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen, "LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*, ed. D. Li and S. Zhou (IEEE, 2023), 206–217.

78. J. T. Liang, C. Yang, and B. A. Myers, "A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, ed. A. Roychoudhury and M. Storey (IEEE/ACM, 2024), 1–13.

79. Q. Ma, T. Wu, and K. Koedinger, "Is AI the Better Programming Partner? Human-Human Pair Programming vs. Human-AI Pair Programming," *arXiv* (2023).

80. J.-Y. Yao, K.-P. Ning, Z.-H. Liu, M.-N. Ning, and L. Yuan, "LLM Lies: Hallucinations Are Not Bugs, but Features as Adversarial Examples," *arXiv* (2023).

81. X. Dong, Y. Wang, P. S. Yu, and J. Caverlee, "Disclosure and Mitigation of Gender Bias in LLMs," *arXiv* (2024).