What Is Generative AI Good For? Introduction to the Special Issue on Generative AI in Software Engineering

Viktoria Stray[a,b], Geir Kjetil Hanssen[b], Astri Barbala[b,c], Darja Šmite[d], Klaas-Jan Stol[e]

[a]*University of Oslo, Oslo, Norway*
[b]*SINTEF, Norway*
[c]*University of Galway, Ireland*
[d]*Blekinge Institute of Technology, Sweden*
[e]*University College Cork, Ireland*

Abstract

A major question that can be asked of any new major technology or innovation is: what is it good for? For this special issue, we invited manuscripts that answer exactly that question in the context of Generative AI and Software Engineering. We received 33 submission, which underwent a rigorous peer review process. This process led to inclusion of 13 manuscripts, which we organized according to McGrath's Group Task typology in this editorial. In doing so, we acknowledge that not all tasks are equal, and we demonstrate the breadth of tasks that GenAI can assist in. This set of curated articles provides a variety of interesting applications and studies of GenAI technology. We conclude this editorial with an outlook on the future.

1. Introduction

Generative AI (GenAI) has become a major theme of great interest to both academic researchers as well as professionals in the software industry. GenAI refers to a category of tools and algorithms that can create new output based on an extensive set of inputs (training data). Current GenAI tools that are highly relevant to software engineering (SE) include GitHub Copilot and ChatGPT, as they can generate source code given a prompt, which serves as a specification. This appears to be a natural fit for SE, given the field has traditionally had a strong focus on requirements, which are frequently expressed in natural language. GenAI tools can be used in all phases of the software development lifecycle (whichever process is followed), including code generation, testing, and debugging. The possibilities and challenges seem almost limitless. While many studies of GenAI have started to appear, we are only scratching the surface of the impact of this new type of technology.

This special issue sought to present a collection of papers that present different aspects of, and perspectives on, GenAI in an SE context, and as such to represent a snapshot of research as a collection of curated papers that can provide a useful entry point for both researchers entering the field, software practitioners, and software managers. We received a total of 33 manuscripts; each of which was screened, and 25 were subjected to a rigorous peer review process involving three reviewers per manuscript. Out of those, we accepted 13 manuscripts.

The focus of this special issue is on a wide range of questions on how GenAI can be used in SE, but also on the challenges and consequences of adopting this technology in SE. As such, the special issue seeks to provide a holistic overview of GenAI in SE. While GenAI will have no doubt a major impact on SE practice, there are numerous open questions that have remained unanswered. For example:

- What benefits does GenAI bring to organizations and individual users?
- How does GenAI influence SE practices?
- What factors might play a role in deciding whether or not to adopt GenAI technology?
- What challenges does adopting GenAI bring?
- What do developers think of GenAI?
- Are GenAI tools useful for specific tasks like code translation and trace generation of modeling operations?
- Can GenAI tools be used to identify security bug reports?
- Can GenAI technology be used to generate fixes for software vulnerabilities?
- How should developers choose from multiple GenAI-generated code solutions?
- What GenAI technology perform the best given certain criteria, such as performance and response time?

The articles in this special issue answer variants of these questions. Each of these questions assumes that GenAI technologies are used for specific tasks, but it should be clear that not all tasks are equal. In the next section, we adopt a task taxonomy from the group psychology literature that defines eight task types, and organize the articles included in this special issue by task type.

2. What is Generative AI Good For? An Overview of the Articles in this Special Issue

Generative AI is heralded as representing a major paradigm shift in virtually any aspect of our lives. GenAI will change the way we live and do things. GenAI technology, currently primarily in the form of Large Language Models (LLMs) and Code Language Models (CLMs), are now increasingly used to perform tasks, and LLMs have been suggested to be 'team members' to achieve tasks. For example, LLMs have been used as co-authors on papers [1].

To better understand what GenAI technology is good for, it is worthwhile analyzing what tasks GenAI might be good for. The psychology literature has long focused on effective groups as "task performance systems" and in that work there has been a focus on differentiating among tasks; not all tasks are the same, and therefore experiments will yield different results, depending on the task at hand. One useful task taxonomy was proposed by McGrath [2]; despite being over 40 years old, it appears to be surprisingly relevant today. Since GenAI agents are now increasingly seen as potential group members (or assistants), we adopt McGrath's task taxonomy, to better understand what types of tasks GenAI has been used for. The taxonomy (see Figure 1), which McGrath has referred to as the "task circumplex," defines eight task types organized in four main types of processes: Generating (alternatives), Choosing (alternatives), Negotiating, and Executing [2]. We briefly discuss these as a foundation to understand how GenAI has been used in the articles included in this special issue. The four main processes, and the eight task types embedded within these processes, are assumed to happen within a group of people. Instead of people, we shall refer to group members as actors, recognizing that GenAI technology is now frequently seen as an independent 'agent', and that some humans within groups will be replaced by GenAI actors; the processes therefore take place in a group that consists of both humans and GenAI agents.

In the remainder of this section, we discuss the four main processes and the eight task types. It is important to keep in mind that these eight task types form a continuum, rather than distinct categories with sharp boundaries. Some tasks could fit either of two adjacent task types. Some of the studies consider multiple task types; in most cases, we discuss only one of those. Notwithstanding, we argue that even an awareness of the notion of different task types to analyze and reason about different tasks that are performed with GenAI technology helps to gain insights and theorize about the future of GenAI in software engineering tasks. We position the 13 articles included in this special issue along this continuum, noting that some of these articles could fit several of the task type categories. This mapping illustrates the variety of tasks for which GenAI technology has been used, ranging from generation of solutions to coding problems to decision tasks related to organizational adoption of GenAI technology.

2.1. Quadrant I: Generate Tasks

So-called 'Generate' tasks are collaborative. The goal is not to determine a single best answer or to evaluate contributions of different inputs, but rather to generate alternatives. Two types of generate tasks can be distinguished [2]. Type 1 represents *Planning* tasks, which are action-oriented (hence its position on the right-hand side of the task circumplex); these are tasks that identify some steps (actions) to achieve a goal. Type 2 represents *Creativity* tasks; these are less action-oriented, but rather rely on mental or cognitive effort (hence its position on the left-hand side of the task circumplex).

An example of a Type 1 Task is to generate use-cases for effective use of LLMs in startups. This is one of the activities performed by Thea Ahlgren, Helene Sunde, Kai-Kristian Kemell and Anh Nguyen-Duc, in their article **"Assisting Software Startups with LLMs: Effective Prompt Engineering and System Instruction Design"** [3]. Using a Design Science research approach, they first identified a number of use-cases for LLMs in startups. They then experimented with various prompt patterns to optimize LLM responses for these use-cases. Finally, they developed 'StartupGPT,' which is an LLM specifically tailored for startups. This solution was evaluated with 25 startup practitioners using multiple methods.

Another example of Type 1 tasks includes the use of LLMs for project plan generation, which is one of 25 use-cases that Kai-Kristian Kemell, Matti Aarikallio, Anh Nugyen-Duc, and Pekka Abrahamsson identified in their article **"Still Just Personal Assistants? – A Multiple Case Study of Generative AI Adoption in Software Organizations"** [4]. In their multi-case study exploration of seven European companies, they identified several benefits such as saving time on tasks, increased productivity, and increased job satisfaction. However, the authors also found several challenges to GenAI adoption in the companies, both on an organizational and an individual level: Where organization-level challenges included data privacy and legislative concerns, individuals highlighted difficulties with prompt engineering and concerns about the accuracy and reliability of AI-generated outputs. Finally, as mentioned, they identified 25 different tasks that employees utilize GenAI tools for; while one of them is a Type 1 task, several other task types can also be identified which we discuss as part of the respective task types below.

An example of a Type 2 Task is to use a Code Language Model (CLM) to generate vulnerability security fixes. This is what Guru Bhandari, Nikola Gavric, and Andrii Shalaginov propose in their article **"Generating Vulnerability Security Fixes with Code Language Models"** [5]. They introduce *PatchLM*, a model that is fine-tuned on code blocks linked to Common Vulnerabilities and Exposures (CVEs), which significantly outperforms existing CLMs such as CodeT5 and CodeLlama in generating both accurate, robust, and relevant fixes. Their findings demonstrate the value of tailoring CLMs for automated program repair [6] and also open new directions for securing software across diverse programming languages.

A second example of a Type 2 Task is the generation of code solutions by GitHub Copilot. In their article **"Don't Settle for the First! How Many GitHub Copilot Solutions Should You Check?"** [7], Julian Oertel, Jil Klünder, and Regina Hebig present a thorough evaluation of GitHub Copilot's effectiveness in assisting software engineers. Using a LeetCode dataset comprising of more than 2,000 coding problems and over 17,000

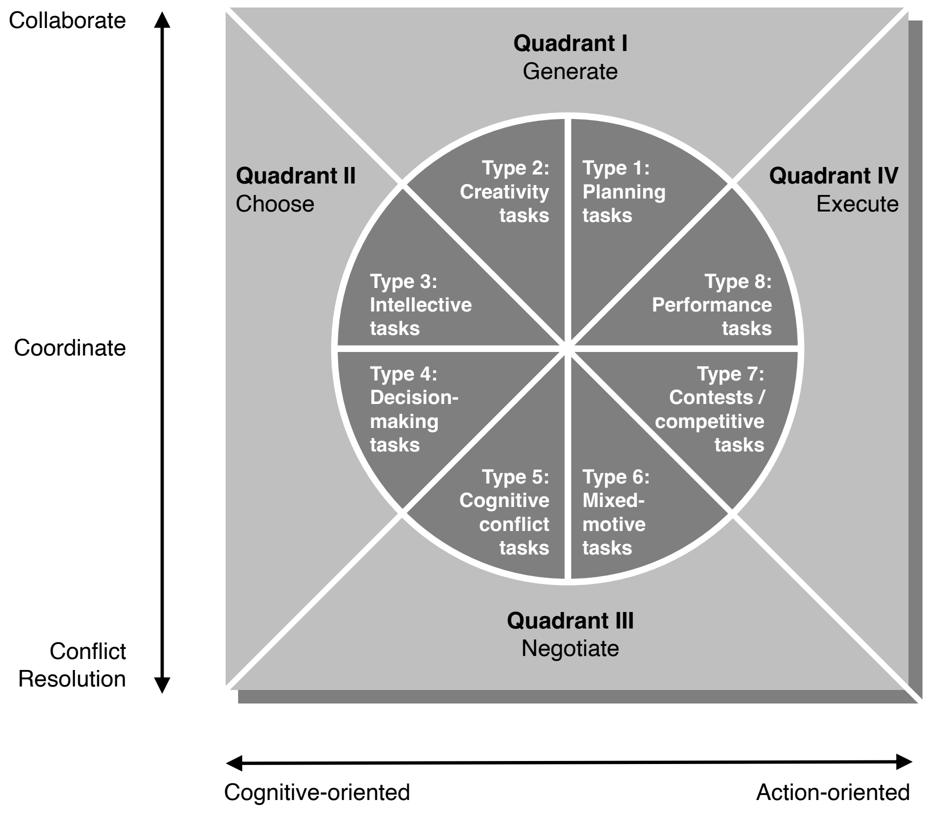

Figure 1: McGrath's Group Task Typology (adapted from McGrath [2])

solutions generated by GitHub Copilot, the authors explore optimal strategies for using GenAI assistance in everyday software development tasks.[1] Unexpectedly, the authors find that the solutions generated by Copilot are not consistently ordered by relevance or correctness. Although selecting the first suggestion may seem convenient, the authors caution that it is less likely to be correct. At the same time, checking ten solutions might be too time-consuming. Based on their findings, the authors thus recommend reviewing at least four-five solutions. Finally, the study results suggest against relying on Copilot's assistance when dealing with uncommon, novel, or difficult problems, as it's suggestions may offer limited value and should be rather used as a source of inspiration and not solution.

We note that several of the 25 SE tasks identified by Kemell et al.'s article (mentioned above) can be classified as Type 2 tasks, for example code generation, comment and documentation generation, and content generation for own product.

2.2. *Quadrant II: Choose Tasks*

Choose tasks are those where a choice is made between alternatives. This quadrant also defines two task types [2]. *Intellective* tasks (Type 3) are those with a demonstrably correct answer, whereas decision-making tasks (Type 4) are those where is no single correct answer, but rather when one alternative may be preferable over another, depending on the criteria and the judgment of the actors involved [2]. The 'correct' answer is determined on the basis on a consensus of actors.

An example of a Type 3 task is the use of LLMs for selecting relevant papers in systematic literature reviews. This is the topic of the article **"Exploring the Use of LLMs for the Selection Phase in Systematic Literature Studies"** by Lukas Thode, Umar Iftikhar, and Daniel Méndez [8]. The authors evaluated five state-of-the-art LLMs on the task of classifying papers as "include" or "exclude" based on their title and abstract, using two completed SLRs as ground truth. Their results were cautiously optimistic. In the best scenarios, a single LLM approach could achieve approximately 98% recall, meaning it identified 98 out of 100 relevant papers. For researchers, this suggests that LLMs could soon serve as useful assistants in literature reviews, as they can dramatically reduce the number of papers that need to be manually screened by filtering out irrelevant studies, while including nearly all the relevant ones.

An example of a Type 4 task is determining how GenAI technology can be leveraged by software developers. This is the topic of the article **"Copiloting the future: How generative AI transforms Software Engineering"** by Leonardo Banh, Florian Holldack, and Gero Strobel [9]. Based on interviews with 18 SE professionals, the authors developed a conceptual framework outlining both the potential benefits and the practical challenges of GenAI integration. On the benefits side, GenAI was found to

[1]It is worth noting that the task of *generating* solutions is, obviously, a Task 2 (Generate), whereas the task of *selecting* a solution is a Choose task, and depending on specifics, can be a Type 3 or Type 4 task, to be discussed below.

Table 1: Overview of articles in this special issue

Task Type	Article	Study Description	Audience and Key findings	Authors' Directions for Future Work
Type 1	Assisting Software Startups with LLMs: Effective Prompt Engineering and System Instruction Design	Design Science. Explore the use of LLMs to support software startups. Development of prompt patterns for a set of specific use-cases; development of StartupGPT, an LLM specifically tailored to startups.	Validated set of use-cases of LLMs to assist startups. Effective prompt patterns and system instructions StartupGPT: an LLM-based startup assistant, as a digital mentor.	Include fine-tuning of StartupGPT to train it on relevant data. Further expansion of the prompts to give further guidelines to the LLM. Explore how to mitigate negative feedback on prompts to ensure responses are better tailored to startups.
Type 1, Type 2, Type 8	Still Just Personal Assistants? – A Multiple Case Study of Generative AI Adoption in Software Organizations.	Field Study. Multiple case study at seven European organizations, focusing on the challenges they face when adopting GenAI for SE; the benefits of GenAI that are achieved, both for the organizations as a whole and individual users; and how GenAI is used and changes SE.	The study identified 25 types of tasks for which the case organizations use GenAI (these vary in type as per the typology). The study identified 12 benefits of using GenAI in SE, and 10 adoption and use challenges, including privacy and legislative concerns.	Investigate the role of job experience on the adoption and perceived usefulness of GenAI. Investigate the use of GenAI in startups, and further study relevance and impact of challenges and benefits identified in this study.
Type 2	Generating Vulnerability Security Fixes utilizing Commit Hunks	Design Science. Proposal of a code language model (CLM), called 'PatchLM,' which is fine-tuned on code blocks linked to Common Vulnerabilities and Exposures (CVEs).	The proposed model PatchLM significantly outperforms existing CLMs in generating both accurate, robust, and relevant fixes.	Integrating tools and human evaluations. Develop bigger and more granular datasets with multi-file changes, configuration updates, and broader context information to train better models. Security-focused fine-tuning using vulnerability-specific datasets. Study a more diverse set of pre-trained CLMs.
Type 2	Don't Settle for the First! How Many GitHub Copilot Solutions Should You Check?	Sample Study. Analysis of 2,025 coding problems, and 17,048 solutions to those problems, generated by GitHub Copilot. (While the generation of solutions is a Type 2 task, the selection of a solution would be a Type 3 task.)	Solutions are not less likely to be correct if they appear at later positions; checking four to five solutions is generally sufficient; difficult and novel problems are unlikely to be solved by GitHub Pilot; the first solution is less likely to be correct; the time needed to check all solutions may be cost-prohibitive.	Industry-based investigations are necessary to evaluate the practical effort developers must invest in reviewing AI-generated solutions.
Type 3	Exploring the Use of LLMs for the Selection Phase in Systematic Literature Studies	Laboratory Experiment. Explore the use of LLMs for the selection phase of systematic literature reviews, evaluating five LLMs on datasets from two published SLRs, using precision and recall metrics.	LLMs improve recall but struggle with precision; few-shot prompts outperform zero-shot; GPT-4 is best but costly. Human oversight remains necessary.	Broaden evaluation to more prompts, models, and SLR datasets; explore other SLR phases (e.g., full-text screening, summarizing). Research hybrid workflows combining LLMs and a human second-reviewer.
Type 4	Copiloting the Future: How Generative AI Transforms Software Engineering	Qualitative study using Grounded Theory techniques, drawing on 18 interviews with professionals to explore opportunities and barriers to adoption.	GenAI enhances productivity and alleviates mental load, particularly in early-stage tasks. However, adoption is slowed by workflow misalignment, trust issues, and prompting effort. The authors propose a conceptual framework for GenAI adoption in SE.	Investigate GenAIs' effect on developer productivity, cognitive load, and data security, and whether over-reliance on GenAI leads to a decline in critical thinking and a degradation of coding skills over time.
Type 5	Trust, Transparency, and Adoption: Social Media Perspectives on Generative AI for Software Engineering	Sample Study. Examines conversations on Twitter/X related to GenAI coding tools / code generation tools (CGT).	Developers applaud CGTs for their ability to improve processes and automation, but also express concerns about ethics, misuse of OSS, and risk of automating complex decision making tasks.	Investigate how CGTs be integrated into SE, considering concerns about bias, IP, and legal compliance. Ethical frameworks for AI code use; transparency in the code generation process, impact of CGTs on workflows and team dynamics.
Type 7	Benchmarking Large Language Models for Automated Labeling: The Case of Issue Report Classification	Laboratory Experiment. Benchmark study to evaluate BERT-like and generative LLMs based on a GitHub dataset.	LLMs demonstrate potential for zero-shot classification, but performance varies considerably across datasets, and they require many computational resources. BERT-like models show more consistent performance, requiring fewer resources.	Fine-tuning of generative LLMs and prompt engineering; investigate how to improve consistency of LLMs across datasets. Explore hybrid approaches that leverage generative and BERT-like models. Explore how to reduce computational overhead of Generative LLMs.

continued on next page

Table 1: Overview of articles in this special issue (continued)

Task Type	Article	Study Description	Audience and Key findings	Authors' Directions for Future Work
Type 7	GPTs are not the Silver Bullet: Performance and Challenges of using GPTs for Security Bug Report Identification	Laboratory Experiment. Explore use of GPT models for identification of security bugs, comparing their performance to traditional ML methods.	GPT models performed poorly to identify security bug reports, with significant variation in performance in recall and accuracy. Use of different prompts also led to considerable differences in recall and accuracy.	Fine-tuning of prompts and improving 'stickiness' of prompt responses; prompt engineering to eliminate positive bias; address prompt token limit; fine-tuning available pre-trained models for security reports; study factors that contribute to correct predictions; investigate automatic data quality improvement; increase availability of high-quality datasets.
Type 7	Assessing Output Reliability and Similarity of Large Language Models in Software Development: A Comparative Case Study Approach.	Laboratory Experiment. Evaluation of reliability and similarity of outputs generated by five LLMs.	The models demonstrate an overall similarity of only 57%, emphasizing that LLMs are not interchangeable and that human oversight is necessary.	Investigate the influence of model architectures and training approaches on performance variations. Analysis of task-specific performance patterns to explain variation in reliability. Analyze how models handle different application domains. Investigate error patterns vis-a-vis architectural and training choices. Investigate impact of prompt engineering on model performance.
Type 7	Translating Code with Large Language Models and Human-in-the-loop feedback	Laboratory Experiment. Proposal of a methodology and metrics for evaluating how Gen AI tools can streamline code translation.	LLMs with an intuitive interface were preferred; ChatGPT generated most reliable code; no clear winner in terms of execution time. Copilot showed how to manage code complexity. Bard was the fastest, and ChatGPt demonstrated highest precision.	Focus on integrated advanced optimization techniques to produce relevant code to address specific tasks. Develop clear guidelines and best practices for secure and effective use.
Type 8	Leveraging Synthetic Trace Generation of Modeling Operations for Intelligent Modeling Assistants Using Large Language Models	Laboratory Experiment. Explore the interplay between traditional MDE and LLMs, and evaluate the ability to train an LLM for model completion based on past operations.	Based on an evaluation in two modeling environments with industry use-cases, LLMs are demonstrated to generate synthetic traces that are comparable to those made by humans.	Consider additional modeling assistants and LLMs that account for long-range temporal dependencies.
Type 8	Using Large Language Models for Multi-Level Commit Message Generation for Large Diffs	Laboratory Experiment. Use of LLMs to generate commit messages for 'long' diffs.	GPT-4o and LLaMA 3.1 70B performed best in generating accurate and relevant commit messages. Commonly used metrics to evaluate (BLEU, METEOR, ROUGE) may not be the most reliable as they do not agree with human evaluators.	More accurate evaluation approaches for commit message generation are needed that balance the benefits of automation and human insight. More comprehensive and robust assessments based on larger datasets.

help streamline software development by assisting with planning, coding, debugging, and routine tasks. It supported early-stage design work by acting as a creative partner when exploring ideas (Type 2 tasks) and helped developers understand unfamiliar code. Participants reported that GenAI reduced development time, eased cognitive load, and helped identify coding errors. They perceived an increase in productivity and felt they had more time for complex and strategic work.

However, the study also identified barriers to adoption, including model limitations, data privacy concerns, misalignment with established workflows, reliability concerns (such as hallucinations and inconsistent outputs), and the effort required to prompt and validate GenAI outputs. Additionally, the authors mention a risk of over-reliance, which can lead to a decline in critical thinking and a degradation of coding skills over time.

2.3. *Quadrant III: Negotiate Tasks*

McGrath positions 'Negotiate' tasks as an extension of 'Choose' tasks, but whereas the former seeks to 'solve' a solution in selecting a specific choice, the latter seeks to 'resolve' [2]. In negotiate tasks, actors who seek a joint solution may disagree and in order to progress may need to compromise. The root of the disagreement can be due to a conflict in viewpoints (or philosophies) (Type 5), or a conflict of interests (Type 6). In Type 5 tasks, parties share the same goal and purpose, and any disagreements are due to *cognitive conflicts*. An example in economics is Keynesian vs. Monetarian policies: both Keynesians and Monetarists share the same goal of a thriving economy, but their approaches towards that goal differ based at a cognitive (or 'principled') level.

The question of whether or not to adopt GenAI tools is one that we would classify as a decision-making task (Type 4). Clearly, there is no single correct answer, but the decision ultimately relies on a number of factors. These factors may include concerns, criticisms, expectations, and preferences. Manaal Basha and Gema Rodríguez-Pérez focused exactly on this. They studied Twitter discourse on the use of Generative AI for software engineering in their article **"Trust, Transparency, and Adoption: Social Media Perspectives on Generative AI for Software Engineering"** [10]. In seeking to understand sentiments surrounding developers' use of Code Generation Tools (CGTs), they collected and analyzed data from approximately 90,000 tweets. Despite many users praising CGTs for streamlining development and automating routine tasks, the study found that sentiments were often divided. Developers also expressed concerns about ethical integrity, legal compliance, the risks of automating complex decision-making, as well as bias in training data – particularly in sensitive industries.

A second type of task in the Negotiate quadrant is called mixed-motive tasks (Type 6). Whereas all previously discussed task types (Types 1 to 5) assume that all actors *"want the same outcome for the group—the 'correct' answer"* [2], Type 6 tasks may have an outcome that is optimal for one or more actor, but not for the others. A classic example of this is the Prisoners' Dilemma. This type of tasks shifts the focus of *cognitive* conflict to a "struggle for pay-offs"; that is, the outcome will be determined based on actors' differing interests. For example, the question of whether or not it should be legal to train LLMs on copyrighted material has been a much-discussed question that depends on a conflict of interests [11, 12].[2] Clearly, it is in the interest of the creators of GenAI technology, such as OpenAI, to use copyrighted material, and they do so under the premise of 'fair use' [13]. Understandably, however, creators of original materials (authors, songwriters, etc.) have a very different set of interests.

None of the studies in this special issue describe tasks of Type 6. Within SE, a good example of a Type 6 task, as it relates to GenAI, is deciding on its use in education. At one level, SE educators and students have the same goal, namely that students successfully graduate (which would make it a Type 5 task). However, at a different level, the degree certificate is only a proxy outcome; the real goal of educators is that students acquire expertise in all aspects of SE, whereas some students (those who do not shy away from cheating) may only focus on receiving the degree; the "payoff' for cheating (without being caught) is *"avoiding work, achieving higher grades and/or realizing less stress"* [14]. For that reason, whether or not to allow the use of GenAI is a *mixed-motive* task that involves at least two different parties with potentially differing goals (making it a Type 6 task). Several papers have been published that discuss this and related concerns [15, 16].

2.4. Quadrant IV: Execute Tasks

What McGrath calls 'Execute' (or 'Perform') tasks focus on performing and implementation of plans [2]. This quadrant differentiates so-called 'contests' or 'battles' (Type 7) from performances that strive to meet a certain standard (Type 8). In a contest, the task will lead to a 'winner' and a 'loser.' In a performance, on the other hand, the task is not a competition, but an activity to *"meet standards of excellence (or, sometimes, standards of 'sufficiency'"* [2, p.65]). The goal in performance tasks is not to win, but to achieve a certain goal. Several of the articles in the special issue describe tasks that fall within this quadrant.

An example of a Type 7 task (a competition or contest) is described by Dae-Kyoo Kim and Hua Ming, who examined reliability and similarity of the outputs of several LLMs in their article **"Assessing Output Reliability and Similarity of Large Language Models in Software Development: A Comparative Case Study Approach"** [17]. Following a structured approach, Kim and Ming evaluated the reliability and similarity of outputs from five prominent LLMs: ChatGPT, Claude, Copilot, Gemini, and Meta. A key finding is that the models demonstrated an overall similarity of only 57%, which emphasizes that these models are not interchangeable and that human oversight is necessary.

Differences in reliability suggest the need for calibrated human oversight, especially in complex design decisions and emerging technology domains. The authors propose a hybrid approach.

Another example of a Type 7 task is explored in the article **"Benchmarking Large Language Models for Automated Labeling: The Case of Issue Report Classification"** [18], by Giuseppe Colavito, Filippo Lanubile, and Nicole Novielli. They present a comprehensive benchmark study on LLMs for automatic labeling and issue classification. The authors evaluated 22 generative LLMs and two BERT-like encoder models across two GitHub datasets, one manually validated and the other using crowd-sourced labels. Their results show that, while generative models perform competitively in zero-shot settings, they often struggle with label consistency and output formatting. In contrast, fine-tuned encoder-based models, especially SETFIT, achieve more stable and accurate results with significantly lower inference time and hardware requirements. The study highlights that data quality and label clarity critically impact performance, particularly for generative models. Despite the flexibility of LLMs, practical deployment is challenged by high computational cost and the need for response post-processing. As such, the authors argue that BERT-style models remain preferable in many real-world scenarios, especially when modest labeling effort is acceptable. However, development within this field is not done, and to support future research, they have shared a fine-tuned sentence-transformer model and a complete replication package for their study.

A third example of a Type 7 task is presented in a study by Gabriele Dario De Siano, Anna Rita Fasolino, Giancarlo Sperlí, and Andrea Vignali, who developed a methodology and metrics for evaluating how Generative AI tools can streamline code translation in their article **"Translating Code with Large Language Models and Human-in-the-loop feedback"** [19]. Where previous studies have focused on evaluating translation quality, the authors here centralizes the human-AI interaction to investigate how individuals use ChatGPT, Google Bard, and GitHub Copilot to translate from code written in query languages to code written in framework-specific code languages, specifically

[2]https://www.spinellis.gr/blog/20250626/

focused on SQL dialects and PySpark. Their study highlights how iterative tool interaction, human refinement, and context-aware adaptation contribute to both efficiency and code quality, offering practical insights for integrating AI into development workflows.

Another example of a Type 7 task is classifying security bug reports. In **"GPTs are not the Silver Bullet: Performance and Challenges of using GPTs for Security Bug Report Identification"** [20], Horácio França, Katerina Goseva-Popstojanova, César Teixeira, and Nuno Laranjeiro present an empirical evaluation of four state-of-the-art, off-the-shelf GPT models for classifying security bug reports. Drawing on seven datasets from open-source projects, they explore how model selection, prompt formulation, and dataset characteristics impact classification performance. The authors compare GPT-based approaches (using zero-shot learning and three different prompts) against classical machine learning models like Support Vector Machine, Random Forest, and Logistic Regression.

Contrary to the expectations surrounding GPTs' versatility, the study finds that GPT models consistently underperform traditional ML classifiers in terms of precision, F-score, and accuracy. While GPTs achieve high recall in some cases, this is often at the expense of precision, resulting in many false positives. Moreover, results vary substantially across datasets and prompts, highlighting prompt sensitivity and the influence of data quality and class imbalance.

The authors identify several challenges, including token limits, prompt bias, and the lack of fine-tuning, as critical barriers to effective GPT use in security bug report identification. They conclude that current GPT models are not yet viable replacements for classic methods and offer detailed recommendations to guide future research toward possibly overcoming these limitations.

One example of a *performance task* (Type 8) is the work of Vittoriano Muttillo, Claudio di Sipio, Riccardo Rubei, and Luca Berardinelli, who explored the interplay between traditional model-driven engineering (MDE) and LLMs. In their article **"Leveraging Synthetic Trace Generation of Modeling Operations for Intelligent Modeling Assistants Using Large Language Models"** [21], they propose a conceptual model that comprises a modeling environment in which modeling operations are recorded. Model completion is achieved by a modeling assistant that is trained on those past operations. The feasibility of this approach in practice is evaluated in two modeling environments, with use cases from industry. The results show that the LLMs used in this study are able to generate synthetic traces that are comparable to human ones.

Another example of a Task Type 8 is the generation of commit messages, which are subsequently evaluated to a certain standard. This is addressed in the article **"Using Large Language Models for Multi-Level Commit Message Generation for Large Diffs"** by Abhishek Kumar, Sandhya Sankar, Partha Pratim Das and Partha Pratim Chakrabarti [22]. Using several different LLMs, they investigate how effective these LLMs are in generating commit messages, in particular with 'larger' diffs, given that much work on automated commit message generation has focused on diffs of only up to 200 tokens. Kumar et al. found that LLaMA 3.1 70B emerged as the best model based on common evaluation metrics (BLEU, METEOR, ROUGE-L, and CIDEr metrics). Their study also finds that some LLMs generate more accurate and relevant messages than traditional baselines, offering evidence that LLMs may be very effective to support software engineering tasks. Finally, they also find that human evaluation is a more reliable approach than a metric-based one (using ROUGE and METEOR, for example), which are less likely to capture the nuanced quality of commit messages.

Table 1 presents a summary of the 13 articles in this special issue, capturing the Task Type [2], research strategy [23], target audience and key findings, and a summary of directions for future work as proposed by the authors.

3. Future Outlook

The 13 papers in this special issue demonstrate a wide variety of studies of GenAI in SE, ranging from comparative studies with "traditional" ML techniques, explorations of the performance of GenAI technologies, to qualitative studies investigating opinions and perceptions of software professionals. Across these articles, we can observe a number of recurring themes that can be tied to the Task Circumplex (Figure 1), which we believe merit further investigation.

- Human-AI collaboration: How will GenAI reshape the role of the human knowledge worker, and the organization as a whole? One example is AI assistants in agile software development [24]. The Copenhagen Manifesto has argued that "Generative AI in Software Engineering Must Be Human-Centered" [25], a sentiment also found in other communities [26] (e.g., collaborative tasks, including Generate tasks: Type 1 and 2).

- To what extent can we trust the correctness of solutions, decisions, and advice provided by GenAI technology in making decisions and selecting alternatives, or the code solutions generated by GenAI (e.g., Type 3, and 4)?

- How do we address issues related to trust, ethics, privacy, and legislation? What if AI-generated outputs "disagree" with human norms, judgment, or legislation? Where do we draw the line of trust in GenAI? (e.g., Negotiate tasks: Type 5 and 6)?

- Studying a variety of quality attributes of GenAI technology: what is the performance, quality, and reliability of AI-generated vs. human-generated outputs? For which tasks does GenAI perform better than a human knowledge worker (Execute task: Type 7) or than a certain standard of excellence (Execute task: Type 8)?

Much research of GenAI technology in SE revolves around the question: How well can GenAI (usually: LLM or CLM) perform task X? This type of question is very useful, and the impact of such research studies can be readily applied. However, we observe two major challenges. First, such studies will be quickly outdated with the release of newer versions of GenAI technology. The next version of a given LLM may perform dramatically better — or worse — invalidating the results of such studies quickly. This makes much of this research short-lived.

Second, we observe a dramatic growth in publications that focus on applying GenAI in SE activities, and the result is an enormous number of empirical findings. Unfortunately, very little of this work is driven by or focused on generating theory. There are a few exceptions, such as Russo's recent work on the adoption of GenAI in software engineering [27]. Future empirical SE studies could benefit from a more theory-centric approach [28, 29], and from keeping the goals of evidence-based software engineering [30] in mind to serve the software industry. These are [31]:

- Develop a deep understanding: enrich our understanding of techniques, practices of GenAI technology in SE;
- Lead to insights that are practical and meaningful: identify outcome measures that are practical and meaningful to practitioners;
- Support assessment and evaluation: conduct studies that generate convincing evidence to help determine whether techniques, practices, and approaches actually work in practice;
- Support decision making: conduct studies that help software professionals and organizations to make better decisions.

This special issue is part of the "conversation" that we, as a scholarly community, have. This conversation has many participants—even the most AI-skeptical researchers within the community cannot deny or ignore the huge influence that GenAI technology has on SE research and practice. For this conversation to be productive, we must carefully "listen" to different perspectives and provide structure—McGrath's task circumplex is one attempt to find a theoretical structure among latent patterns of research of GenAI in SE. Other useful theoretical frameworks exist; one particularly useful framework that has been used to by Storey et al. is McLuhan's Tetrad [32]. This framework helps to ask "interesting" questions; for example: how does GenAI technology enhance SE practice? How does it make practices and technologies obsolete? How does it reverse common practice, and what past practices might it retrieve? Similarly, we can identify "interesting" questions by challenging and problematizing assumptions that of current literature, rather than simply "spotting gaps," i.e., identifying research questions that have remained unanswered — some things are simply not interesting [33, 34].

Apart from an increasing number of primary studies that investigate GenAI technology in SE, we can also observe many review and "roadmap" papers. Such papers help address the problem of not being able to see the forest for the trees. At the same time, it is imperative that, as a field, we remain thoughtful, considerate, and critical in proposing future research questions, aiming for contributions that endure beyond the next publication. The SE research community has the capacity to generate knowledge that truly shapes both our knowledge, understanding and practice.

This special issue marks only the beginning of the conversation around GenAI in SE. Information and Software Technology as one of the leading journals will feature other special issues on GenAI: upcoming special issues are focused on "Engineering GenAI-enabled Software Systems," edited by He (Jason) Zhang, Shang Gao, and Xiaofeng Wang, and "Next-Generation Model-Based Software Engineering with Foundation Models," edited by Claudio Di Sipio, Silvia Abrahao, Fabio Palomba, and Martin Weyssow. Let the conversation continue!

Acknowledgements

We express our sincere gratitude to the committed reviewers who devoted their time and valuable input to this special issue. We are grateful to Special Content Editor Dr Jeff Carver, who provided prompt and frequent assistance as needed. Finally, we are grateful to the authors who entrusted us with their research and responded carefully to all reviewer feedback. This research was supported by the Research Council of Norway through grants 321477 and 355691.

References

[1] Q. Zhang, C. Gao, D. Chen, Y. Huang, Y. Huang, Z. Sun, S. Zhang, W. Li, Z. Fu, Y. Wan, L. Sun, LLM-as-a-coauthor: Can mixed human-written and machine-generated text be detected?, in: K. Duh, H. Gomez, S. Bethard (Eds.), Findings of the Association for Computational Linguistics: NAACL 2024, Association for Computational Linguistics, Mexico City, Mexico, 2024, pp. 409–436. doi:10.18653/v1/2024.findings-naacl.29.

[2] J. E. McGrath, Groups: Interaction and performance, Prentice Hall, 1984.

[3] T. L. Ahlgren, H. F. Sunde, K.-K. Kemell, A. Nguyen-Duc, Assisting early-stage software startups with llms: Effective prompt engineering and system instruction design, Information and Software Technology 187 (2025) 107832. doi:10.1016/j.infsof.2025.107832.

[4] K.-K. Kemell, M. Saarikallio, A. Nguyen-Duc, P. Abrahamsson, Still just personal assistants?–a multiple case study of generative ai adoption in software organizations, Information and Software Technology 186 (2025) 107805. doi:10.1016/j.infsof.2025.107805.

[5] G. Bhandari, N. Gavric, A. Shalaginov, Generating vulnerability security fixes with code language models, Information and Software Technology 185 (2025) 107786. doi:10.1016/j.infsof.2025.107786.

[6] C. Le Goues, M. Pradel, A. Roychoudhury, Automated program repair, Communications of the ACM 62 (12) (2019) 56–65. doi:10.1145/3318162.

[7] J. Oertel, J. Klünder, R. Hebig, Don't settle for the first! how many github copilot solutions should you check?, Information and Software Technology 183 (2025) 107737. doi:10.1016/j.infsof.2025.107737.

[8] L. Thode, U. Iftikhar, D. Mendez, Exploring the use of llms for the selection phase in systematic literature studies, Information and Software Technology 184 (2025) 107757. doi:10.1016/j.infsof.2025.107757.

[9] L. Banh, F. Holldack, G. Strobel, Copiloting the future: How generative ai transforms software engineering, Information and Software Technology 183 (2025) 107751. doi:10.1016/j.infsof.2025.107751.

[10] M. Basha, G. Rodríguez-Pérez, Trust, transparency, and adoption in generative ai for software engineering: insights from twitter discourse, Information and Software Technology 186 (2025) 107804. doi:10.1016/j.infsof.2025.107804.

[11] N. Rahman, E. Santacana, Beyond fair use: Legal risk evaluation for training LLMs on copyrighted text, in: Proceedings of the 1st Workshop on Generative AI and Law (GenLaw '23), colocated with the 40th

International Conference on Machine Learning (ICML), 2023, https://blog.genlaw.org/2023-workshop.html.

[12] S. C. Lightstone, Train or restrain? using international perspectives to inform the american fair use analysis of copyright in generative artificial intelligence training, Northwestern Journal of International Law & Business 44 (3) (2024) 471.

[13] P. Samuelson, Generative AI meets copyright, Science 381 (6654) (2023) 158–161. doi:10.1126/science.adi0656.

[14] P. A. Hutton, Understanding student cheating and what educators can do about it, College Teaching 54 (1) (2006) 171–176. doi:10.3200/CTCH.54.1.171-176.

[15] V. D. Kirova, C. S. Ku, J. R. Laracy, 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 V. 1, 2024, pp. 666–672. doi:10.1145/3626252.3630927.

[16] I. Joshi, R. Budhiraja, P. D. Tanna, L. Jain, M. Deshpande, A. Srivastava, S. Rallapalli, H. D. Akolekar, J. S. Challa, D. Kumar, "with great power comes great responsibility!": Student and instructor perspectives on the influence of LLMs on undergraduate engineering education, arXiv preprint arXiv:2309.10694 (2023).

[17] D.-K. Kim, H. Ming, Assessing output reliability and similarity of large language models in software development: A comparative case study approach, Information and Software Technology 185 (2025) 107787. doi:10.1016/j.infsof.2025.107787.

[18] G. Colavito, F. Lanubile, N. Novielli, Benchmarking large language models for automated labeling: The case of issue report classification, Information and Software Technology 184 (2025) 107758. doi:10.1016/j.infsof.2025.107758.

[19] G. D. De Siano, A. R. Fasolino, G. Sperlí, A. Vignali, Translating code with large language models and human-in-the-loop feedback, Information and Software Technology 186 (2025) 107785. doi:10.1016/j.infsof.2025.107785.

[20] H. L. França, K. Goseva-Popstojanova, C. Teixeira, N. Laranjeiro, Gpts are not the silver bullet: Performance and challenges of using gpts for security bug report identification, Information and Software Technology 185 (2025) 107778. doi:10.1016/j.infsof.2025.107778.

[21] V. Muttillo, C. Di Sipio, R. Rubei, L. Berardinelli, Leveraging synthetic trace generation of modeling operations for intelligent modeling assistants using large language models, Information and Software Technology 186 (2025) 107806. doi:10.1016/j.infsof.2025.107806.

[22] A. Kumar, S. Sankar, P. P. Das, P. P. Chakrabarti, Using large language models for multi-level commit message generation for large diffs, Information and Software Technology 187 (2025) 107831. doi:10.1016/j.infsof.2025.107831.

[23] K.-J. Stol, B. Fitzgerald, The ABC of software engineering research, ACM Transactions on Software Engineering and Methodology 27 (3) (2018) 11. doi:10.1145/3241743.

[24] B. Cabrero-Daniel, T. Herda, V. Pichler, M. Eder, Exploring human-ai collaboration in agile: Customised llm meeting assistants, in: International Conference on Agile Software Development, Springer Nature Switzerland Cham, 2024, pp. 163–178. doi:10.1007/978-3-031-61154-411.

[25] D. Russo, S. Baltes, N. van Berkel, P. Avgeriou, F. Calefato, B. Cabrero-Daniel, G. Catolino, J. Cito, N. Ernst, T. Fritz, H. Hata, R. Holmes, M. Izadi, F. Khomh, M. B. Kjærgaard, G. Liebel, A. L. Lafuente, S. Lambiase, W. Maalej, G. Murphy, N. B. Moe, G. O'Brien, E. Paja, M. Pezzè, J. S. Persson, R. Prikladnicki, P. Ralph, M. Robillard, T. R. Silva, K.-J. Stol, M.-A. Storey, V. Stray, P. Tell, C. Treude, B. Vasilescu, Generative AI in software engineering must be human-centered: The Copenhagen Manifesto, Journal of Systems and Software (2024). doi:10.1016/j.jss.2024.112115.

[26] D. Wang, E. Churchill, P. Maes, X. Fan, B. Shneiderman, Y. Shi, Q. Wang, From human-human collaboration to human-AI collaboration: Designing AI systems that can work together with people, in: Extended abstracts of the 2020 CHI conference on human factors in computing systems, 2020, pp. 1–6. doi:10.1145/3334480.3381069.

[27] D. Russo, Navigating the complexity of generative AI adoption in software engineering, ACM Transactions on Software Engineering and Methodology 33 (5) (2024) 1–50. doi:10.1145/3652154.

[28] V. Stray, R. Hoda, M. Paasivaara, V. Lenarduzzi, D. Mendez, Theories in agile software development: Past, present, and future, Information and Software Technology 152 (2022) 107058. doi:10.1016/j.infsof.2022.107058.

[29] K.-J. Stol, M. Goedicke, I. Jacobson, Introduction to the special section—general theories of software engineering: New advances and implications for research, Information and Software Technology 70 (2016) 176–180. doi:10.1016/j.infsof.2015.07.010.

[30] B. A. Kitchenham, T. Dyba, M. Jorgensen, Evidence-based software engineering, in: Proceedings of the 26th International Conference on Software Engineering, IEEE, 2004, pp. 273–281. doi:10.1109/ICSE.2004.1317449.

[31] S. Beecham, D. Bowes, K.-J. Stol, Introduction to the ease 2016 special section: Evidence-based software engineering: Past, present, and future, Information and Software Technology 89 (2017) 14–18. doi:10.1016/j.infsof.2017.05.002.

[32] M.-A. Storey, D. Russo, N. Novielli, T. Kobayashi, D. Wang, A disruptive research playbook for studying disruptive innovations, ACM Transactions on Software Engineering and Methodology 33 (8) (2024) 1–29. doi:10.1145/3678172.

[33] K. H. Rolland, B. Fitzgerald, T. Dingsøyr, K.-J. Stol, Acrobats and safety nets: problematizing large-scale agile software development, ACM Transactions on Software Engineering and Methodology 33 (2) (2023) 1–45. doi:10.1145/3617169.

[34] M. Alvesson, J. Sandberg, Constructing research questions: Doing interesting research, 2nd Edition, SAGE Publications Ltd, 2024.