International Journal of Innovative Research in Education, Technology & Social Strategies
p-ISSN: 2465-7298 | e-ISSN: 2467-8163
Volume 12, Number 1 May, 2025

IJIRETSS

Automated Code Generation & AI Tools: From Ideas to Applications

[1]Joni Turunen & [2]Aleksanteri Fagerholm
[1&2]Lappeenranta- Lahti University of Technology, LUT - Finland

Article DOI: 10.48028/iiprds/ijiretss.v12.i1.06

Abstract

The Technology Report discusses how AI is changing software development, with a focus on automated code generation. It examines the latest AI research, tools, and technologies transforming the field. The study evaluates the pros and cons of AI-driven code generation and its impact on efficiency, productivity, and software quality.

Keywords: *Artificial Intelligence (AI), Technology, ChatGPT, Copilot*

Corresponding Author: Joni Turunen

First Published: https://www.researchgate.net/publication/376174254_Technology_Report_From_Ideas_to_Applications

Background to the Study

In an era where digital transformation influences every corner of our lives, touching even the very healthcare solutions keeping us alive [3], the way we develop software and applications has also seen radical shifts. The advent of artificial intelligence (AI) is changing how we interact with applications and how we create them. This technology report delves into a futuristic yet increasingly plausible avenue: the use of AI in the automatic generation of applications and other products too [4] directly from user prompts varying from ideas to Software Requirement Specification (SRS) documents, user stories and other textual input that transform ideas to applications. In this report, we seek to answer, '*How can AI help create applications without direct human involvement in code-writing?*' and who do these new possibilities affect?

Motivation

Since late 2022, after OpenAI released the GPT3 large language model (LLM) to the public in the form of the ChatGPT, there has been increasing attention to AI implementations in public. Discussions of hypothetical Artificial General Intelligence, or AGI for short, with the ability of an AI system to understand, learn and apply knowledge in a wide variety of contexts, much like a human can, have spurred as the AI-leap has taken over the digital world. Outside of academia, attempts to implement AGI-like systems started appearing on GitHub soon after OpenAI revealed its GPT3 model via API to the public, most notably BabyAGI, AutoGPT [5,6] and ChatGPT plugins. LLM-based systems taking an iterative approach that can use external tools to grasp larger tasks have become a new paradigm for generative code generation called AI agents [7].

While the LLMs and how we use them keep evolving and constraints like the size of query context are relieved by increasing the sequence length or parallel prompting [8,9,10], several efforts exist to create projects that build on these new capabilities. Pekka Abrahamson's AI Lab at Tampere University is researching how to prompt these models and utilize them for Software Engineering [11]. These early attempts and research efforts show that the LLMs can be wielded to act in different software development roles, from developer to product owner to tester. Similar approaches have been seen from the likes of Microsoft, releasing AutoGen as open source to the public and implementing AI agents to solve complex software engineering tasks to produce applications [12]. These early projects have shown that trivial programs like a snake game can easily be achieved in minutes. AI/ML has been receiving huge interest in the business world and in academic research efforts. New AI/ML research papers are being published with high volume [13]. AI/ML is causing a new disruption in digitalization that touches all aspects of work, education and industry [7]. From our (authors') work experience, the possible performance, velocity, and monetary opportunities are strong drivers for AI implementations in the software engineering field. Learning and utilizing this new, rapidly evolving technology can mean a competitive edge in the business domain and later evolve into a requirement to be competitive. Finland and other EU countries have needed more developers in the past few years. This bottleneck in the Software industry is holding us back [14], and we believe AI-generated code could help alleviate this problem.

In this technology report, we display our research findings considering the current state of AI technologies considering the code generation from "Ideas to applications". Our technology report strives to find the frameworks and methods software engineers and companies could utilized to build software solutions faster, more efficiently and more securely. By understanding the current possibilities, we can better analyse the impacts this new AI leap is causing and understand what the future may hold for software engineering.

Scope

While this topic is expansive, this report will focus on the current capabilities of AI in automated code generation, drawing from real-world examples and cutting-edge research. Furthermore, while it is essential to stay present, we will also evaluate future capabilities, making informed predictions about the technology's direction and potential effects in the software engineering domain by examining the latest research papers and predictions. This technology report is aimed at decision-makers, managers and developers in software engineering. The report focuses on AI-generated applications from the viewpoint of a software company operating from the EU, building on top of available platforms, APIs and existing LLMs. Producing one's own LLM is outside the scope of this report, as it is very demanding in terms of computational resources and required capital.

We recognize that the AI leap has touched many fields and is changing the world for all digital content. While the socioeconomic impacts and ethics regarding the use of AI are essential topics, our report views the subject from a technological viewpoint, focusing on the impacts and benefits of software companies and businesses. While the capabilities of AI and the generative code outcomes are the priority, the impacts on software engineering are of interest as well. Therefore, Impacts and benefits are considered by studying the Software Engineering stakeholders. Our scope is to evaluate possible impacts on the job market, company performance and profitability and how the work of software engineers is expected to transform.

Data Collection and Processing Methodology

During the beginning of this task, we agreed on our data collection and processing methodology. Our idea was to read research around our topic, post the most influential links to our digital work environment and build our scope from there. After our initial literature gathering the scope became clearer and then we moved into processing of these papers. The processing methodologies included studying and harvesting the parts of these papers which supported our research scope. We built our hypothesis and technical review around these studies and reflected our findings with each other. From the findings throughout the research, we were able to build the structure to this report and from there it was efficient to split between the areas where both of us worked. The writing of this report was partly challenging, since new relevant material was published weekly. Due to this a lot of the material lacked peer review's, making them part of grey literature. In addition to this, the research wasn't able to keep up with the ever-changing landscape of AI, this resulted in deprecated research data.

Understanding recent research and development

The landscape around software development has been evolving and reshaping quite rapidly amid the introduction of automated code generation and AI tools. This rapid growth had an explosive ignition with the release of ChatGPT to the public audience, where the general interest around GenAI spiked in a very short time span. This chapter aims to lay out the timeline around the recent research and development made in this domain by looking at the research and innovations which have sculped the sector in the past couple of years. Since the GenAI tools which have sparked the initial interest of software developers were introduced in the late 2021, this chapter scopes the timeline from that point forward.

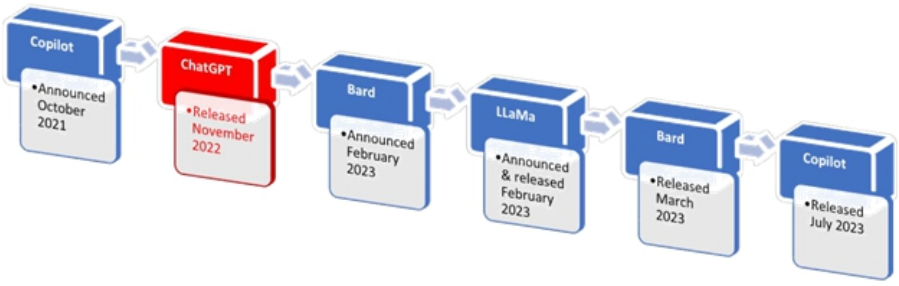

Figure 1: GenAI related releases on a timeline

During the past few years GenAI has not only been able to establish foothold in the software development field, but it has also restructured on software engineers approach coding and code creation. This will possibly mean that future software development will increasingly be incorporated with auto-generated code and the capabilities to work with such tools needs to be taught within these roles and in the curriculum of current students [15]. This change isn't just a new trend, but it portrays a shift that is happening on how certain parts of software is planned, created, and maintained. With the tools like ChatGPT and Copilot the capabilities for automated code generation has greatly improved. If we look at Figure 1, we can roughly see the release timeline for different LLM's which have gained the attention from the public. It all started with the announcement of GitHub's Copilot which introduced an AI leveraged pair programmer tool. The announcement was able to ignite the discussion on how LLM could help developers to do tasks which Copilot could provide the code for. The Copilot had a limited beta testing environment, so it wasn't released out for all public to test. A little while after during November of 2022, OpenAI released their main product ChatGPT-3 out to the public which generated significant interest from wide range of various fields, attracting developers and other users to its userbase. The launch of this was huge and it became the fastest growing website during its time, by being able to generate 100 million users in just two months [16]. Introduction of ChatGPT-3 to the public was a catalyst accelerating the interest of public crowds, and it wasn't just its users, but investors jumped in as well and the race for publishing AI research and products increased significantly.

Following the release of ChatGPT, other companies had to announce their products as well, to convey the trust of investors. In the early stages of year of 2023, additional companies rose to

public stage to fight for space in this new emerging market. From the current big tech companies, Google and Facebook revealed their products to the market with the hopes of gaining attraction from the public. These releases were able to offer public and researchers different LLM's which clearly differed from the GPT model 3. For example, LLaMA got recognition on outperforming the GPT model 3, even though the model was 10 times smaller compared to the OpenAI's product. The research around LLamMa also demonstrates that solely using publicly accessible data can produce very competitive models without the need for utilizing proprietary datasets [17]. Research around Bard highlighted its access to internet but pointed out that where GPT had limitations in biases around data, Bard itself is limited with biases of the internet [18]. The releases of all these products in such a short timespan creates positive race towards the development of LLM's, where the competition and co-existence of multiple models feed diversity and innovation in their approaches. This also provides researchers multiple different angles to understand the benefits, obstacles and advancements which this industry needs to understand. Additionally, going forward this landscape needs to balance between open-source policies and proprietary advancements, while challenging its users to navigate through unknown data and messaging about collective progress.

Tools and technologies related to AI code generation

In this chapter, we will give a general overview of different technologies and tools related to code generation using AI. By introducing the vast possibilities of different tools and techniques, the report aims to broaden the view of software engineers, managers, product owners and decision-makers.

AI assistants for software developers

GitHub copilot was introduced to a technical preview in July 2021 and was later launched as a commercial product. CoPilot stirred opinions as the public started using it concerning code quality, misuse, and general code security [19]. CoPilot integrates with the developer's IDE, like Visual Studio Code or IntelliJ. CoPilot continues to evolve and has added new features, including a chat-like interface to ask for explanations and code fixes for specific program code segments in the IDE. While GitHub CoPilot powered by Microsoft and OpenAI is a paid service, AWS released Code Whisperer in July 2023, which takes a similar approach, integrates with the IDE tools and is a free service.

The use of AI assistants like CoPilot in software engineering tasks shows productivity increases via a 55.8% increase in completion time and hints at a better success rate on programming tasks [20]. AI assistants are limited by the context window restrictions that does not always catch important code segments to produce accurate results during inference.

Fine-tuning LLMs

LoRA has received significant attention for fine-tuning methods [31]. In addition to the techniques presented, one of the more recent methods, Quanitized LoRA or QLoRA [32], looks promising for fine-tuning Open Source LLMs on precise domain knowledge that requires software engineering tasks. QLoRA can reduce the computational requirements of

the fine-tuning process, lowering the costs of fine-tuning pre-trained models to suit specific tasks. While community-developed models are becoming more available via services/communities such as Hugging Face, they may pose threats and risks to software companies due to the unclear fine-tuning data and methods to generate the new model. Within the Open Source LLMs, we have also noticed uncensored, fine-tuned chat models that provide answers to unethical questions.

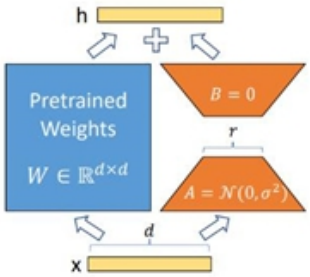

Figure 2: LoRA fine-tuning principle for adding new parameters (weight AB) [24]

Agent Approach and Managing Context Windows

Most current LLMs provide a limited context window for inference. For code generation purposes and software engineering tasks in particular, this presents undesired restrictions. Tasks related to code refactoring or debugging and understanding existing code structure and functions with context window limitations reduce the accuracy on large code bases. To manage this issue, an iterative approach to code generation equips AI with the possibility to re-evaluate its earlier decisions and test outcomes. This feedback loop for AI allows earlier mistakes to be fixed and errors corrected. One of the first publicly available implementations to achieve these was AutoGPT [6]. AutoGPT takes in several LLM APIs and uses those to leverage more recent factual knowledge or to generate different output formats like voice via ElevenLabs AI API. Using AutoGPT solves some of the issues, but prompt management becomes an issue and requires 'context switching' for the session to the chosen LLM API.

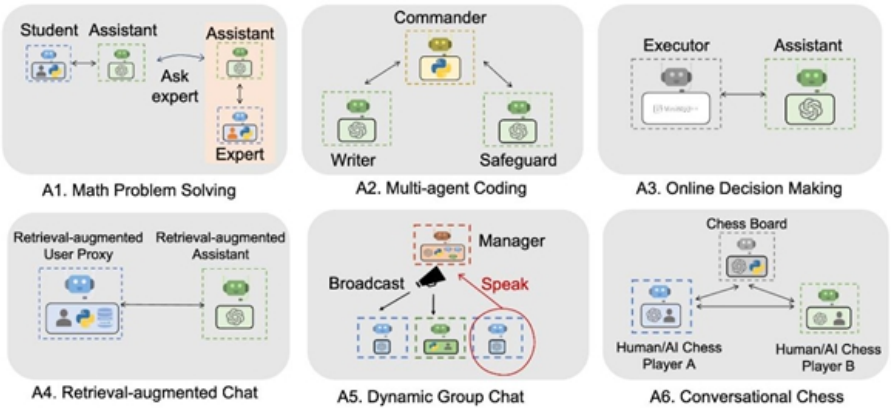

Figure 3: Use case demonstrations for AutoGen [8]

Microsoft introduced the AutoGen framework in August 2023, which enables multiple AI agents with customisable roles and the ability to engage in agent-to-agent conversations. With this approach, we can define different software engineering roles such as developer, tester, product owner, business analyst, requirements engineer and others to engage in the software project and the conversation [12]. AutoGen allows an iterative approach where the agents can detect flaws and mistakes and refactor the code in alignment to match the original request. While AutoGen conversational agents help to handle larger contexts and can chop tasks to move towards the goal in smaller steps, the context of larger code bases is still challenging to handle.

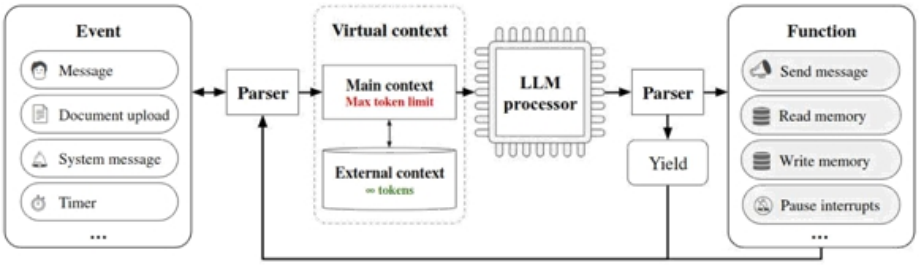

Figure 4: MemGPT Virtual Context management via Functions to mimic infinite context length [29]

MemGPT was introduced on October 12th, 2023, and it introduces a virtual context management system that resembles operating systems handling slow and fast memory in the form of random-access memory and persistent data volumes. MemGPT achieves this by injecting the context window with instructions to handle memory and use function-calling capabilities in recent LLMs. Applying MemGPT creates an illusion of an infinite context window [34]. Available MemGPT libraries allow us to set up AutoGen agents using MemGPT for LLM APIs, including GPT-3.5 and GPT-4 models, but also for open-source LLMs like Mistral 7B.

Knowing the Advantages and Disadvantages

The fast adaptation of new technologies will reveal advantages and disadvantages to its users. These new evolving tools offer their users new ways to increase efficiency, streamline development processes and harness new protocols around problem-solving. As promising as this sounds, with this new technology, we gain both positive and negative outcomes. Since LLM research is a relatively new study subject, it is rather challenging to keep track of the current emerging advantages and disadvantages these researchers have proven. Moreover, with these negatives comes the need to control things [35]. At this point, the new research lacks real-life use cases with empirical data, so its claims give us guidelines on what these advantages and disadvantages might consist of.

With the disadvantages, our focus is on software developers who need to become more familiar with what happens under the hood of these LLM products. It may not be clear to the public that LLMs do not serve as repositories that store explicit knowledge; instead, they are

dynamic systems that analyses and learn from different data patterns. This means that when these people unfamiliar with the subject are reading the text that these machines generate, they cannot distinguish which of the content provided has been made by humans and which additions around the content are machine-generated. This might not sound like a big thing, but the aftereffect generates the problems around this matter. By being unable to distinguish between these two, the user of GenAI is faced with ethical dilemmas that they might be unaware of. The text they obtain might contain, for example, plagiarism, violation of privacy, misinformation, and biased data towards certain demographic groups. There haven't been proper solutions for this disadvantage yet, but the legislation is vastly evolving, and different public figures have brought transparency on what is being done on behalf of new laws around GenAI. The user group, which is more familiar with LLM's behaviour and has a good understanding of the domain where they operate, have a better starting point to overcome this obstacle. They can overcome the "black-box" disadvantage by utilising LLMs more as thinking tools while keeping in mind that they should have capabilities and means of validating the outputs they provide. This solution remains effective as long as these machine-generated texts are not found in different open-source materials where experts in these domains cannot distinguish the text between human-made and machine-generated [36].

In addition to the user groups, disadvantages and technical constraints around LLM's also exist. For example, the Context Window is a restriction in which there is a workaround, but the setup of the LLMs and environments is still complicated and expensive. Research about the outcome quality is lacking. The context is only partially captured on a large code basis, and the LLMs can generate confidently sub-optimal answers. With our experience using these systems, especially GPT4 and CoPilot, AI can provide accurate and fast answers if the program is small in size and appropriately structured. Adjusting and giving more context to the LLM can improve the answer, but after 2-3 interactions with the intent to get better results from LLM, the accuracy withers, and the same time and effort is used better in actual implementation by the developer.

AutoGen systems should include human intervention, validation, and approval checks from the AI; otherwise, the process can easily slip down the wrong track. As described, the current state only has good accuracy with human intervention. From the 'ideas to applications' point-of-view, the current state cannot provide applications from a general prompt describing non-naive applications. For the advantages, we can start by categorising different user groups on how they could utilise these tools in the software engineering field. For requirement engineering, GenAI can offer different development patterns where stakeholders can assess the accuracy of various software requirements; although the LLM-generated requirements might contain certain abstractive approaches, studies have shown them also to be coherent and comprehensible. In order to say that software developers gain an advantage from the utilisation of GenAI, it is a little bit more complex situation. Throughout history, they have incorporated different automated tools that have made their work easier and faster, but this time, it might require some additional training for these models to work alongside developers. At this point, the intuition to utilise these tools beneficially might derive from the background or seniority of the developer. They can, for example, use GitHub Copilot to generate unit tests

from the code they have shown to the model. However, even though the model provides working output, it is up to the developer to confirm that the solution is applicable. For the more junior developers or students, these GenAI tools could operate as chatbots and help them with different coding tasks outside the classroom. In their content, they would explain what errors the student might have made in their solutions. Also, in addition to this, the model could assess the student's code and provide answers about the state of the solution and which improvements the student could make to it. In software project management, GenAI could help by providing better resource allocation, performing cost estimations, organising requirements according to the technical strategy, and planning sprints using the resources at hand [21].

Conclusion and Findings

In this chapter, we summaries the research findings and identify the key beneficiaries and negatively impacted stakeholders of automatic code generation and AI tools.

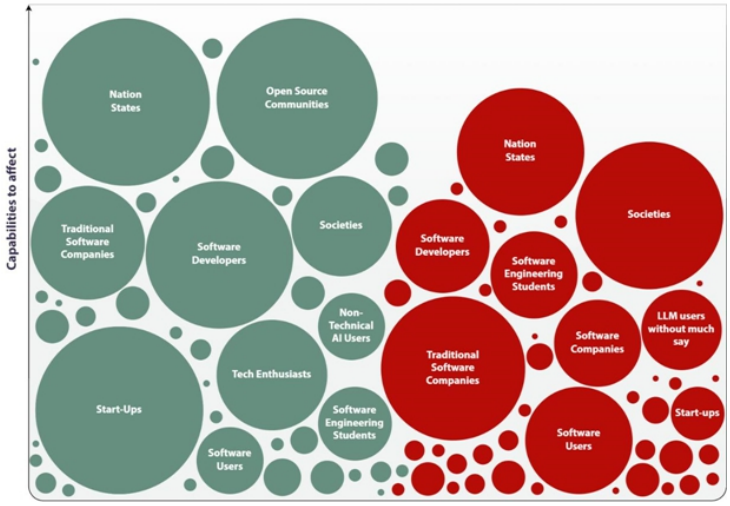

Figure 5: Stakeholder analysis of GenAI positive and negative impacts

Research has shown that automatic code generation and AI tools have the potential to bring significant benefits to various stakeholders. These tools can enhance the development experience for developers by enabling them to start new projects quickly, avoid wasting time on boilerplate code, or help in refactoring and improving the performance of their code quickly. Additionally, these tools provide software company owners with the opportunity to increase their velocity and productivity. By automating specific tasks, developers can focus on more complex and creative aspects of their work. Automatic code generation and AI tools can bring significant benefits to customers. By providing more options in the market, they have the potential to improve the overall quality of software development. With the assistance of AI, developers can deliver more robust and efficient solutions that meet customer requirements effectively.

Beneficiaries
The beneficiaries of automatic code generation and AI tools are manifold. Firstly, software company owners can greatly benefit from these tools as they can significantly increase the velocity and productivity of their companies. By automating specific tasks, companies can accelerate their development process and improve overall efficiency.

Secondly, customers can enjoy a broader range of options in the software market due to the use of these tools. Furthermore, automatic code generation and AI tools have the potential to enhance the quality of the software being developed, resulting in more reliable and efficient solutions.

Lastly, developers can benefit from these tools by improving their development experience. With automatic code generation and AI tools, developers can easily refactor their code and optimise its performance, enabling them to focus on more complex and creative aspects of their work.

Negative impacts
The increasing use of automatic code generation and AI tools may negatively impact traditional code developers who are reluctant to engage GenAI tools and methods. They may need help regarding market demand and competitiveness as these tools can automate tasks that manual developers traditionally perform. Entry-level job descriptions might vanish, and software engineering tasks become more demanding, making it harder for junior-level programmers and graduates to enter the market. Software development and the IT industry, in general, are known as fast-moving fields, but due to the AI leap and recent developments, the need to adopt new methods and learn new skills is faster than usual.

There may also be negative impacts on job markets concerning software engineering roles if conversational AI agents can scale to the challenges of enterprise-level code bases. Start-ups and knowledgeable individuals can jump-start new businesses and challenge monolithic traditional companies that cannot respond to the change fast enough. New competition may cause unpredictable challenges for many of these old software companies.

The digital divide (DD) between actors who can invest in AI-capable hardware for training and fine-tuning LLMs and utilise existing LLMs to their full potential and those who can only subscribe to services will grow the gap larger. We expect the DD between data holders, LLM providers and the users to grow. Also, the existing DD between different digitalisation levels in nation-states is going to be affected. Pioneering countries can wield AI implementations in their defence forces and military endeavours, while other less advanced nations face new risks and threats to their sovereignty.

Societies that are affected by all digital services are slowly starting to take advantage of AI technologies and may see growth in their GDP but also challenges in job markets and are required to invest in education. Concerns about data protection and privacy are highlighted in general, not only because of software development and the use of AI in applications. Concerns

of immaterial rights for software companies are also a problem and need regulation and governance from public administrations that currently need to catch up.

Final Thoughts

While automatic code generation and AI tools have their advantages, it is essential to consider their limitations and potential shortcomings. These tools may only partially replace human intervention and validation; at least, that is the case now. Without proper human oversight, there is a risk of the process deviating from its intended track. Moreover, the accuracy and capabilities of these tools vary as the models tend to produce creative outcomes by default, making them unpredictable for software developers, managers and companies. In the software industry, automatic code generation and AI tools have immense potential to benefit software company owners, customers, and developers. However, it is crucial to consider the negative impact these tools might have on all the beneficiaries. Also, we must consider the limitations of such tools to ensure their successful implementation and utilization in the future.

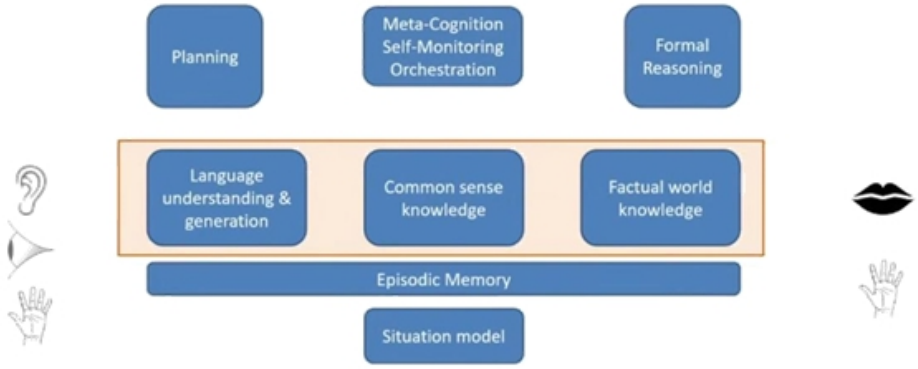

Figure 6: Multi-modal AI system dissociation between LLMs language capabilities and factual world knowledge [30]

Research agendas are being written and published as of the time of writing this tech report, defining future research needs [21]. New information and knowledge are being introduced in grey literature services such as arXiv.org. Research is moving fast, and new findings surface every week. Some researchers suggest a need for multi-modal AI systems where factual world knowledge would be detached from the language capabilities, and the current LLM architecture would be further dissected into modules to reduce the computational and monetary costs of training LLMs in the future [37].

Experts in the field are making bold guesses about when the next steps towards Artificial General Intelligence (AGI) will be taken. Estimations of achieving AGI vary from 12-24 months. The time will tell when the next leap happens. In general, whatever the practical implementations are, we expect them to be integrated and enhance current automation solutions, like robotic process automation [38]. Businesses are racing to adopt AI

technologies into their products, causing a dilution of the term AI. Some of these solutions may be wrappers on OpenAI's API, while some are simple algorithms. Continuous investment is needed in these companies developing GenAI for the next leap to happen. Some of the investment initiatives stem from the research made around these language models, which in turn may shape the grey literature as organizations try to acquire the investments. For us to better understand the state of these language models, peer-reviewed research is essential.

References

[1] Brown, T. B., et al. (2020). *Language models are few-shot learners,* ArXiv.Org, https://doi.org/10.48550/arxiv.2005.14165.

[2] Vaswani, A. et al. (2023), *Attention is all you need*, ArXiv.Org, https://doi.org/10.48550/arxiv.1706.03762.

[3] Usmani, U. A., et al. (2023). *Artificial intelligence applications in healthcare*, Proceedings of Eighth International Congress on Information and Communication Technology. ICICT 2023. Lecture Notes in Networks and Systems, vol. 694, edited by X.S. Yang et al., Springer, https://doi.org/10.1007/978-981-99-3091-3_89.

[4] Ghoreishi, M., & Happonen, A. (2020). New promises AI Brings into circular economy accelerated product design: A Review on Supporting Literature." E3S Web of Conferences, 158, ,pp. 1-10, doi: 10.1051/e3sconf/202015806002.

[5] Nakajima, Y. & Babyagi, G. (2023), *https://github.com/yoheinakajima/babyagi.*

[6]Richards, T. B. (2023*). Autogpt." github*, https://github.com/Significant Gravitas/AutoGP

[7] Benaich, N. (2023). *State of AI report,* Air street capital, 2023, www.stateof.ai/.

[8] Ding, J. et al. (2023*). LongNet: Scaling transformers to 1,000,000,000 Tokens,* arXiv.org, https://doi.org/10.48550/arxiv.2307.02486.

[9] Open AI. (2023). *Models. open AI documentation, Open AI, 2023*, platform.openai.com/docs/models. Accessed 3 Nov..

[10] Ratner, N. et al. (2023). Parallel context windows for large language models." arXiv.org, https://doi.org/10.48550/arxiv.2212.10947.

[11] Abrahamson, P. (2023). *Generative AI, large language models and ChatGPT: What should the software companies do now?"* Lahti Software Day, 29 Aug., Lahti, Finland.

[12] Wu, Q. et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via multi-agent conversation, https://doi.org/10.48550/arxiv.2308.08155.

[13] Krenn, M. et al. (2022). *Predicting the future of AI with AI: High-quality link prediction in an exponentially growing knowledge network*." arXiv.org,

[14] Roiha, R. (2023*). Software finland "An overview to the state and the future of the Finnish tech companies,* Lahti Software Day, 29 Aug., Lahti, Finland.

[14] Becker, B. A., et al (2023), *Programming Is Hard – Or at least it used to Be: educational opportunities and challenges of ai code generation*, https://dl.acm.org/, https://dl.acm.org/doi/pdf/10.1145/3545945.3569759

[16] Gustafsson, S. (2023). *Generative language models for automated programming Feedback,* diva-portal.org, https://www.diva-portal.org/smash/get/diva2:1784396/FULLTEXT01.pdf

[17] Azhar, F, et al. (2023). *LLaMA: Open and efficient foundation language models,* arXiv.org, https://arxiv.org/pdf/2302.13971.pdf.

[18] Singh, S. K. et al. (2023). *Chat GPT & google Bard AI: A review*, 2023 International Conference on IoT, Communication and Automation Technology (ICICAT), IEEE, pp.1–6, https://doi.org/10.1109/ICICAT57735.2023.10263706.

[19] Moradi, D. A. et al. (2023). GitHub copilot ai pair programmer: Asset or liability?, *The Journal of Systems and Software, 203,* 111734–https://doi.org/10.1016/j.jss.2023.111734.

[20] Sida, P. et al. (2023). *The impact of AI on developer productivity: Evidence from GitHub copilot,* arXiv.org,, https://doi.org/10.48550/arxiv.2302.06590.

[21] Nguyen-Duc, A. et al. (2023). Generative Artificial Intelligence for Software Engineering - A Research Agenda." ArXiv.Org, , https://doi.org/10.48550/arxiv.2310.18648.

[22] Devlin, J. et al. (2019). BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding."arXiv.org, https://doi.org/10.48550/arxiv.1810.04805.

[23] Liu, Y. et al. (2019). RoBERTa: A robustly optimized BERT pretraining approach, arXiv.org, https://doi.org/10.48550/arxiv.1907.11692.

[24] Raffel, C. et al. (2023). *Exploring the limits of transfer learning with a unified text-to-Text transformer.*, arXiv.org, https://doi.org/10.48550/arxiv.1910.10683.

[25] Yang, Z. et al. (2020). *XLNet: Generalized autoregressive pretraining for language understanding,* arXiv.org, , https://doi.org/10.48550/arxiv.1906.08237.

[26] Singh, M. et al. (2023). *CodeFusion: A pre-trained diffusion model for code generation*, arXiv.org, , https://doi.org/10.48550/arxiv.2310.17680.

[27] Rozière, B. et al. (2023). *Code Llama: Open foundation models for code*, arXiv.org, https://doi.org/10.48550/arxiv.2308.12950.

[28] Touvron, H. et al. (2023). *Llama 2: Open foundation and fine-tuned chat models,* arXiv.org, https://doi.org/10.48550/arxiv.2307.09288.

[29] Chen, M. et al. (2021). *Evaluating large language models trained on code*, arXiv.org, https://doi.org/10.48550/arxiv.2107.03374.

[30] Jiang, A. Q., et al. (2023). *Mistral 7B,* arXiv.org, https://doi.org/10.48550/arxiv.2310.06825.

[31] Hu, E. J., et al. (2021). *LoRA: Low-Rank adaptation of large language models,* arXiv.org, https://doi.org/10.48550/arxiv.2106.09685.

[32] Dettmers, T. et al. (2023). *QLoRA: Efficient finetuning of quantized LLMs*, arXiv.org, https://doi.org/10.48550/arxiv.2305.14314

[34] Packer, C. et al (2023). "MemGPT: Towards LLMs as Operating Systems." arXiv.org, https://doi.org/10.48550/arxiv.2310.08560.

[35] Usmani, U. A., et al. (2023). *Enhancing artificial intelligence control mechanisms: Current practices, Real life applications and future views,* Lecture Notes in Networks and Systems, 559, , pp. 287-306. doi: 10.1007/978-3-031-18461-1_19.

[36] Strasser, A. (2023). On pitfalls (and advantages) of sophisticated Large Language models, arXiv.org, , https://arxiv.org/ftp/arxiv/papers/2303/2303.17511.pdf

[37] Mahowald, K, et al. (2023). *Dissociating language and thought in large language models: A cognitive perspective*, arXiv.org, https://doi.org/10.48550/arxiv.2301.066