

City Research Online

City St George's, University of London

Citation: Belbachir, A., Blake, M. B., Dong, J. S., Dustdar, S., Ebert, C., Ghezzi, C., Harel, D., Helal, S., Hu, C., Huang, L., et al (2026). Towards socio-cyber-physical systems: a software perspective amid AI breakthroughs. Science China Information Sciences, 69(9), 193101. doi: 10.1007/s11432-026-5037-1

This is the published version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/38617/

Link to published version: https://doi.org/10.1007/s11432-026-5037-1

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

SCIENCE CHINA
Information Sciences

• POSITION PAPER •

September 2026, Vol. 69, Iss. 9, 193101:1–193101:37
https://doi.org/10.1007/s11432-026-5037-1

Towards socio-cyber-physical systems: a software perspective amid AI breakthroughs

Assia BELBACHIR[10], M. Brian BLAKE[6], Jin Song DONG[9], Schahram DUSTDAR[18], Christof EBERT[22], Carlo GHEZZI[11], David HAREL[23], Sumi HELAL[19], Chunming HU[3], Linpeng HUANG[12], Hans-Arno JACOBSEN[20], Changjun JIANG[15], Zhi JIN[24], Jialin LI[9], Xuanzhe LIU[1], Jian LÜ[2*], Xiaoxing MA[2], Assaf MARRON[23], Hong MEI[1*], Bashar NUSEIBEH[14], Xin PENG[5], Yuanchun SHI[16], Huaimin WANG[8], Ji WANG[8], Jianmin WANG[17], Danny WEYNS[7], James WOODCOCK[21], Tao XIE[1], Hong Henry XU[13], Mengfei YANG[4], Jianwei YIN[25] & Naijun ZHAN[1]

[1]*Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, School of Computer Science, Peking University, Beijing 100871, China*
[2]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China*
[3]*State Key Laboratory of Complex & Critical Software Environment, School of Computer Science and Engineering, Beihang University, Beijing 100191, China*
[4]*China Academy of Space Technology, Beijing 100094, China*
[5]*College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200433, China*
[6]*Georgia State University, Atlanta GA 30303, USA*
[7]*Department of Computer Science and Media Technology, Linnaeus University, Växjö 351 95, Sweden*
[8]*State Key Laboratory of Complex & Critical Software Environment, College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China*
[9]*School of Computing, National University of Singapore, Singapore 119077, Singapore*
[10]*NORCE Norwegian Research Centre, Grimstad 4879, Norway*
[11]*Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan 20133, Italy*
[12]*School of Computer Science, Shanghai Jiao Tong University, Shanghai 200240, China*
[13]*Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong 999077, China*
[14]*School of Computing and Communications, The Open University, Milton Keynes MK7 6AA, UK*
[15]*School of Computer Science and Technology, Tongji University, Shanghai 200092, China*
[16]*Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*
[17]*School of Software, Tsinghua University, Beijing 100084, China*
[18]*Distributed Systems Group, Technische Universität Wien, Vienna 1040, Austria*
[19]*Department of Computer and Information Science and Engineering, University of Florida, Gainesville FL 32611, USA*
[20]*Department of Electrical and Computer Engineering, University of Toronto, Toronto ON M5S 3G4, Canada*
[21]*Department of Computer Science, University of York, York YO10 5GH, UK*
[22]*Vector Consulting Services, Stuttgart 70499, Germany*
[23]*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 7610001, Israel*
[24]*School of Computer Science, Wuhan University, Wuhan 430072, China*
[25]*College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China*

Received 18 April 2026/Revised 17 June 2026/Accepted 6 July 2026/Published online 1 September 2026

Abstract Software technology is undergoing a paradigm shift driven by two converging trends. First, the scope of software responsibility has expanded significantly: as "software-defined everything" becomes a reality, software has evolved into the integration core of socio-cyber-physical systems (SCPSs). Second, the capabilities and development methods of software are being greatly enhanced by recent breakthroughs in artificial intelligence (AI). This article presents perspectives and observations on software engineering in this era of rapid progress. We aim to outline a set of foundational challenges in engineering SCPSs and highlight the need for innovative software solutions that extend beyond AI technologies alone. Specifically, we examine the need for a new paradigm that can address the complexities introduced by SCPSs, which challenge conventional paradigms through the blurring of system boundaries, continuous lifecycle evolution, and the embracing of inherent uncertainty. We highlight new engineering principles of socio-technical co-design, cyber-physical integration, and development-operation convergence, and a knowledge- and data-driven approach to taming uncertainty. Emerging proposals, including digital humanism, agentic SCPS, ubiquitous operating system, and continuous quality assurance, are discussed alongside possible extensions to existing technologies. We then outline key research directions for both runtime support and quality assurance. On the runtime side, we argue for a new generation of software infrastructure for SCPSs, including unified hardware abstractions, scalable and resilient runtime systems, AI-enabled system management, and human-centric operating system

* Corresponding author (email: lj@nju.edu.cn, meih@pku.edu.cn)

© The Author(s) 2026. This article is published with open access at link.springer.com.

primitives. On the assurance side, we highlight the need for new approaches combining unified socio-cyber-physical modeling, specification of both technical and non-technical properties, data-driven simulation and testing, continuous verification, and runtime monitoring under uncertainty. Furthermore, we present specific challenges within key application domains, including intelligent vehicles, smart manufacturing, and smart cities. In doing so, we aim to stimulate discussion within the software engineering community and encourage support from industry and government to address the critical engineering and governance issues inherent to this new generation of systems.

Keywords socio-cyber-physical systems, ubiquitous computing, software engineering, software-defined everything, intelligent software

Citation Belbachir A, Blake M B, Dong J S, et al. Towards socio-cyber-physical systems: a software perspective amid AI breakthroughs. Sci China Inf Sci, 2026, 69(9): 193101, https://doi.org/10.1007/s11432-026-5037-1

1 Introduction

We are entering a transformative era marked by the profound fusion of human society, cyberspace, and the physical world through ubiquitous computing and seamless connectivity [1]. As the boundaries between software, hardware, human users, and their environments continue to dissolve [2], socio-cyber-physical systems (SCPSs) are becoming the new norm in domains such as medical systems and workflows, automation and robotics, production systems and distributed supply chains, and so on. A socio-cyber-physical system is an integrated system that combines IT with physical components and human actors. The system state and the output of such an SCPS depend on physical input variables, human influences, and algorithms that operationalize standards, laws, and other specifications. SCPSs are more than just embedded software and sensors; they are complex ecosystems that blend computational intelligence, physical interaction, and socio-contextual behavior. For example, a connected insulin pump does not just monitor glucose levels and dispense medication, but also must integrate seamlessly with other devices, ensure cybersecurity, adapt to user behavior, and remain safe under all conditions [3].

Developing such systems is demanding from the outset, beginning with the specification of requirements and constraints such as user experience, governance, functional safety, and cybersecurity. Conventional engineering approaches often fall short in addressing the nuanced dependencies and emergent behaviors of SCPS. As development costs soar and risks escalate, the key to mastering these challenges lies in understanding the interplay of the social, cyber, and physical domains—and aligning requirements, design, and test from the earliest stages. Furthermore, the development of an SCPS is never complete—it must dynamically adapt and evolve during its operation to accommodate continual changes in its environment and requirements. While SCPSs have been a focus of research and development for more than a decade [1, 4–8], there is still a lack of systematic engineering methods and technologies for such systems, especially for their software engineering.

At the same time, breakthroughs in artificial intelligence (AI), particularly in deep learning [9] and large language models (LLMs) [10], have significantly enhanced both the capabilities of SCPSs and their development process. The impact of the AI breakthrough on software technology is profound. First, it provides powerful capabilities for sensing and understanding high-dimensional objects, such as images, audio, video, and natural language (NL), which are necessary for SCPSs. Second, advanced LLMs, as a prototype of so-called "general intelligence" [11], can serve as a source of common sense, whose absence is a fundamental reason why traditional software struggles to handle open environments (cf. the frame problem [12]). Third, with an LLM as the code generator, SCPS development can be significantly accelerated, which is crucial for meeting the soaring demand for SCPSs. However, the incorporation of deep learning and LLMs also poses a grand challenge, because their inherent uncertainty compromises the benefits of abstraction and compositionality of software that rely on full certainty.

This article presents a forward-looking perspective on software science and engineering research for SCPSs. Similar to the acatech report edited by Geisberger and Broy [13], we aim to highlight foundational challenges of engineering intelligent SCPSs and to stimulate discussion within the software research community, with a particular focus on recent developments and emerging methodological perspectives. Furthermore, we emphasize that grounding intelligence in real-world SCPS scenarios requires not only advances in AI algorithms and models, but also innovative software methods, techniques, tools, and systems.

The rest of the article is organized into four parts, each discussing a distinct aspect of software technology: Section 2 discusses the need for a new software paradigm for SCPSs and the challenges in the development process; Section 3 focuses on the runtime support; afterwards, Section 4 concentrates on the quality assurance, and Section 5 discusses some specific application domains of SCPS. Finally, we conclude this paper in Section 6. Figure 1 summarizes the issues discussed later.

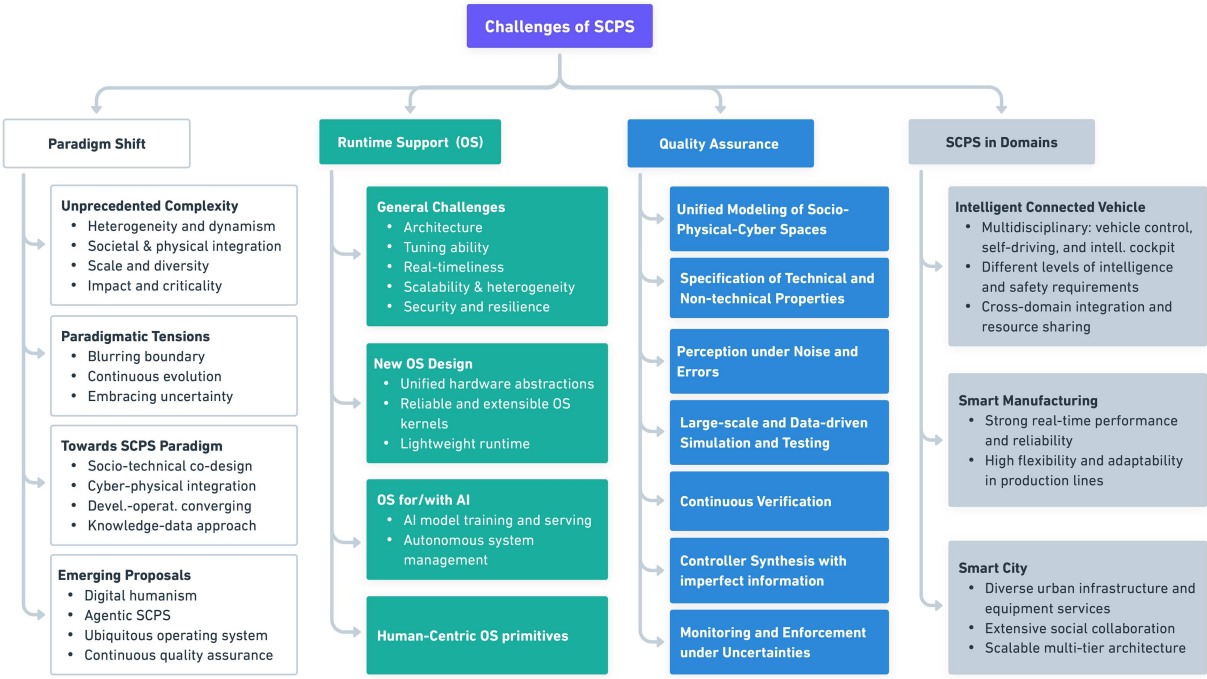


Figure 1 (Color online) Overview of the challenges.

2 Paradigm shift towards SCPS

As Niklaus Wirth once put it [14], "*If we can learn anything from the past, it is that computer science is in essence a methodological subject.*" Before dwelling on the concrete technical and engineering challenges raised by SCPSs, we first discuss some fundamental tensions between conventional software engineering wisdom and the new characteristics of SCPSs from a methodological perspective.

2.1 The concept of SCPS, technically

Technically, we use the term SCPS to refer to a new generation of systems that (1) abstract and encapsulate human/societal, computational, and physical resources through software-defined mechanisms, (2) implement functionality using these resources via knowledge-driven programming and data-driven learning, and (3) serve users in a natural and intelligent manner.

• *Software-defined everything* (SDx) refers not only to the "softwarization" of processes such as sensing, transmission, storage, learning, analysis, decision-making, and control but also to a methodology centered on software for system construction and evolution, enabling effective management of system complexity. In implementation, the software-defined approach employs a software layer that manages heterogeneous resources downward by encapsulating and abstracting operational details while providing programmable, customizable capabilities and services upward through consistent APIs.

• *Human/societal entities* are integral parts of an SCPS. These include users as service recipients and individuals who execute instructions or provide services under incentives, such as delivery personnel. Even system users may participate in operations as human-in-the-loop (HitL) or human-on-the-loop (HotL) entities, such as a driver taking over control in an autonomous driving scenario. Moreover, SCPSs are not entirely disjoint from cyber-physical systems (CPSs); rather, they extend CPSs by elevating human and societal entities from contextual factors or external stakeholders to first-class components. This elevation calls for a socio-technical engineering approach throughout system design, operation, and evolution.

• *Cyber resources* encompass computational machineries, data storage, communication networks, and various software services. SCPSs typically leverage the computational power spanning cloud, edge, and end devices to deliver ubiquitous and seamless services. With the wide adoption of machine learning, training infrastructure and learned models are increasingly critical components of SCPSs.

• *Physical entities* include perceptible physical objects and controllable physical devices within the system. From a software perspective, physical entities are not only sources of sensory data and recipients of control commands but also abstract representations modeled within the software domain via mechanisms such as digital twins. Unified

cross-disciplinary modeling remains an engineering challenge in this context. For example, in autonomous driving systems, vehicle modeling involves not only spatial positioning and path planning but also the kinetics, energy consumption, and cabin state, among others.

• *Intelligence* of an SCPS is not limited to the smart predictions made by artificial neural networks that might be involved (i.e., intelligence by connectionism), but also the rule-based decision-making (i.e., intelligence by symbolism), and the autonomous behavior implemented by the monitor-analyze-plan-execute loop (i.e., intelligence by behaviorism).

2.2 The need for a new paradigm

Software development is a highly creative and complex task. To make it a systematic engineering beyond ad hoc craftsmanship, it is necessary to be *paradigmatic*. A software paradigm is a meta-model shaping how software systems shall be conceived and constructed from the perspective of software engineers [15]. A fundamental question, then, is *how we shall view SCPS paradigmatically*.

2.2.1 *Driving force of software paradigm shifts*

Looking back on the history of the development of software technology, a recurring pattern is that software paradigms shift when new complexities need to be handled, and the complexities emerge together with new generations of computing platforms and new kinds of applications. The shift in the software paradigm led to the restructuring of the software technology stack. The evolution of software paradigms from unstructured to structured, to object-oriented, to component-based, to service-oriented and to the more recent Internetware [16] were all driven by the need to handle *new complexities* brought by the advancement of computing platforms from mainframes to personal computers, to networked computing environments, to the open Internet platform, and also the expansion and enrichment of application domains. For example, the structured paradigm facilitated complexity management in the organization of solutions, while the object-oriented paradigm went further by connecting the problem and solution spaces more closely, reducing the complexity in handling the entangled and evolving requirements of the real world. Every time a software paradigm shifted, software development methods and runtime supporting systems evolved accordingly, as well as software quality assurance.

The development of SCPSs must address a new level of complexity in such systems. The complexity arises not only from a greater number of heterogeneous components and their interactions but also from various new dimensions, such as the following.

• The extent and breadth of integration and dependencies of such systems with respect to many other systems and with the physical environment.

• The fact that these systems both affect/modify and are affected by the very fabric of society and the environment in which they operate.

• The dynamic nature of such systems, with new subsystems appearing and others disappearing, requirements changing dynamically, etc.

• The new scales of diversity induced by such systems—each having many instances, possibly with different versions of software, different ad hoc parameter settings, and operating in different environments.

• The high criticality of them to human well-being, society, the economy, and science.

The impact of this new complexity is profound and revolutionary. This is because it challenges some fundamental principles of conventional software engineering that are crucial for complexity management.

2.2.2 *Blurring the boundary between the machine and the world*

Software development has been understood as the engineering of a *machine* to serve practical purposes in the *world* [17]. Although it has long been emphasized that software engineering should care about not only the machine but also the world, one of the fundamental principles of software development is to define the boundary between the two clearly with a *specification*. The specification, although not always explicitly stated in a formal contract, must be at least well understood and agreed upon. Once we have a specification, the engineering of the machine becomes much simpler, as the machine only needs to exhibit the behavior defined by the specification without worrying about the complexity and chaos of the world. As Michael Jackson put it [17], "*a specification is a staging post on the hazardous journey from a requirement to a program.*"

Unlike conventional software systems, SCPSs are so deeply integrated with the physical and societal world that the separation mentioned above often becomes unnatural and unstable. With the trend of SDx, developing an SCPS involves engineering not only the machine but also (a part of) the world itself. For example, Wi-Fi infrastructure

is usually seen as part of the world, providing networking access, but it can also be software-defined as a sensing device [18] that detects human presence, movement, and even vital signs in an elder care scenario.

More deeply, the reason for this boundary blurring lies in the expanded responsibility that software now bears for the entire real-world system [2]. While software is traditionally responsible only for the information-processing step in the entire real-world business process, for SCPSs, software must act as the ultimate system integrator and take responsibility for realizing the application's business and social value. For instance, consider a food delivery app where every delivery person takes a dual role of both service provider and user. The safety, health, dignity, and fairness of the delivery person must be included in the software requirements alongside the efficient, low-cost fulfillment of service. To meet these requirements, the app and the service platform behind it must be viewed as an integral part of a broader socio-technical construction of the real world, rather than a machine interacting with the world.

The conventional division of machine and world with specification also helps the decomposition of a system into subsystems or components. For each subsystem, we build the machine, and the rest of the system is the world. This principle, when applied recursively, serves as a key tool for managing complexity. However, this hierarchical composition of homogeneous subsystems does not work well for SCPSs either. SCPSs are highly heterogeneous and dynamic. They comprise human, cyber, and physical components, as well as subsystems. The interactions and constraints among these components and subsystems take different forms and vary across topologies. This results in a system with a flexible architecture of systems-of-systems. The components of an SCPS come from different domains and operate under distinct dynamics, obeying distinct underlying laws. The interdisciplinary integration presents a grand challenge, with most aspects remaining unclear or unknown. For example, to support perception in the loop of intelligent control, data engineering and machine learning often become necessary for SCPSs. Nevertheless, the seamless integration of neural and symbolic components to enable intelligent and trustworthy SCPS behaviors remains an open problem.

2.2.3 *Dissolving lifecycle stages into continuous evolution*

Compared to traditional software systems, SCPSs often operate in more open environments and face rapidly changing requirements. This volatility comes from the increasing depth of their physical and societal embedding. Furthermore, many SCPSs, such as air traffic control and smart power grids, need to be non-stop and long-lived. This means that SCPSs should self-react by dynamically adapting their structure and behavior, in order to continually ensure the desired quality of service, which challenges the traditional staged software lifecycle model [19]. Note that the staged lifecycle model is also a means to manage complexity—it divides and conquers the task of software engineering in the temporal dimension.

While agile software methodologies do smooth the iteration of software development, they are still heavily driven by human developers. To be non-stopping and long-living, an SCPS should be capable of *growing*, which means it continuously evolves from its initial version with as little human aid as possible, coping with the everlasting changes in its running environments, dependent resources, and user intentions/preferences by continuously enhancing functionalities and fixing bugs in its original code [20].

Technically, this suggests that the SCPS must be reflexive—it includes not only the functionalities that fulfill its present duties but also "meta-level" facilities for future growth. The facilities may encompass explicit runtime models of users with perceivable intentions and preferences, models of the environment with sensible and predictable dynamics, and models of the system itself with online adaptation and updating capabilities. As AI-assisted automation in software development advances rapidly, integrating development processes into the software itself is becoming more feasible; however, managing its complexity still presents a challenge.

2.2.4 *Shifting from the quest for certainty to embracing uncertainty*

Software is among the most complex artifacts ever built, and it has been incredibly successful. Today, software with tens of millions, sometimes hundreds of millions, of lines of code is installed in every modern phone and car, and people rely on its consistent service without a second thought. The trend of shifting system complexity into software is evident and accelerating. So, what makes software superior to other types of artifacts in managing complexity?

A key factor is that software, as a purely logical artifact, enjoys, in principle, unlimited compositionality and reusability. A program function, regardless of its complexity, can be invoked as a single statement. A software component, assuming its prerequisites are met, guarantees its external behavior precisely and can be combined with other components to create a larger component. This process can be repeated endlessly without any degradation. This abstraction and hierarchical decomposition are well established, not only methodologically but also mathematically [21]. Nevertheless, this unlimited compositionality and free reusability do not come for free. Software must

be implemented as a logical artifact without any uncertainty, which means that during the software development process, all real-world uncertainties must be abstracted away.

However, the inconvenient truth about SCPSs is that we must now acknowledge their inherent uncertainty. The uncertainty comes from the external environment, the physical components, and the humans involved in the system. For example, in an autonomous driving scenario, uncertainties can be caused by external factors such as unpredictable weather conditions and transportation emergencies, by issues with physical components like sensor noise, actuation imprecision, and component degradation, and by imperfect modeling of physical world states and dynamics. Humans are also a major source of uncertainty. For example, consider the unpredictable and illogical behavior of human drivers.

More importantly, as we integrate machine learning components into the SCPS, we must also recognize their inherent uncertainties. First, conceptually, machine learning is essentially a practice of incomplete induction, which is uncertain by nature. The predictions made by a learned model, even with perfect assumptions, are only *probably approximately correct* in theory. Second, practically, randomness, heuristics, and biases are widely employed in the training of machine learning models. All of these can introduce unexplained uncertainties into the behavior of learned models. Finally, for LLMs that are of primary interest today, there is still no theoretical framework for understanding their behavior, and it is widely accepted that hallucination is not a bug but a feature of LLMs [22].

The problem of uncertainty is that it undermines the compositionality and reusability of symbolic software, thereby shaking the foundation of conventional software complexity handling. Interestingly, in his Josiah Willard Gibbs Lecture, Yann LeCun [23] also warns that autoregressive LLMs are doomed because uncertainty explodes exponentially. Possible remedies include confining uncertainty through symbolic guardrails and developing new compositional and reuse mechanisms for uncertain components—both of which go beyond current software engineering practices.

From the discussion above, the complexities of SCPS go beyond traditional software development principles, such as clear boundaries between software and its environments, separation of development and maintenance or evolution, and the pursuit of certainty as a priority. These complexities arise from the revolutionary expansion of SCPS responsibilities beyond conventional software systems and the profound transition to AI-powered system implementation. SCPSs demand a robust and responsive methodology and engineering approach that accounts for social interaction, user behavior, and systemic complexity.

2.3 Towards a new software paradigm for SCPS

An SCPS is not a standalone monolithic system, but rather a system of systems (SoS) [24]—or at least part of one—that is deeply integrated with the real world. From the earlier discussion in Subsection 2.1, it is clear that SCPSs often exhibit general SoS "ABCDE" characteristics, namely, *Autonomy, Belonging, Connectivity, Diversity, and Emergence* [25]. Furthermore, SCPSs are characterized by their embedding in societal and physical contexts. We highlight their characteristic features, paradigm principles, and potential enabling techniques in Figure 2.

2.3.1 *Socio-technical co-design for human-centric computing*

Social integration is a key feature that distinguishes SCPS from other computing systems. An SCPS is a socio-technical construct that encompasses not only technical artifacts but also humans and *institutions* [26]. Institutions are the structures and rules guiding human interactions to achieve collective goals. This feature necessitates a shift from a machine-centric to a human-centric paradigm [1], which in turn requires socio-technical co-design of SCPS.

• *Upholding social and ethical values.* It has long been recognized that, as Tim Berners-Lee [27, p. 124] said, "*technologists cannot simply leave the social and ethical questions to other people, because the technology directly affects these matters.*" However, the urgency of this principle has never been so evident for SCPSs. Society and institutions now depend heavily on SCPSs, making it crucial to ensure security, safety, privacy, fairness, freedom of choice, and societal well-being throughout their entire lifecycle.

• *Socio-technical joint optimization.* Socio-technical co-design has long been proposed to optimize both organizational performance and the quality of working life by simultaneously designing the social and technical aspects of a system [28]. Compared with conventional information/computing systems, SCPSs not only have a greater need for socio-technical co-design due to their deeper social integration, but also greater feasibility thanks to the flexibility enabled by their pervasive, in-depth adoption of SDx and AI.

• *Design for HitL and HotL.* Unlike physical devices and computing components, humans are creative, unreliable, slow, preferring natural interaction, intolerant of repetition, and driven by motivation. As a result, SCPSs with HitL or HotL require careful consideration of these factors during their development. Interesting proposals such as *human machine team* [29] and *the point of no return* [30] are emerging.

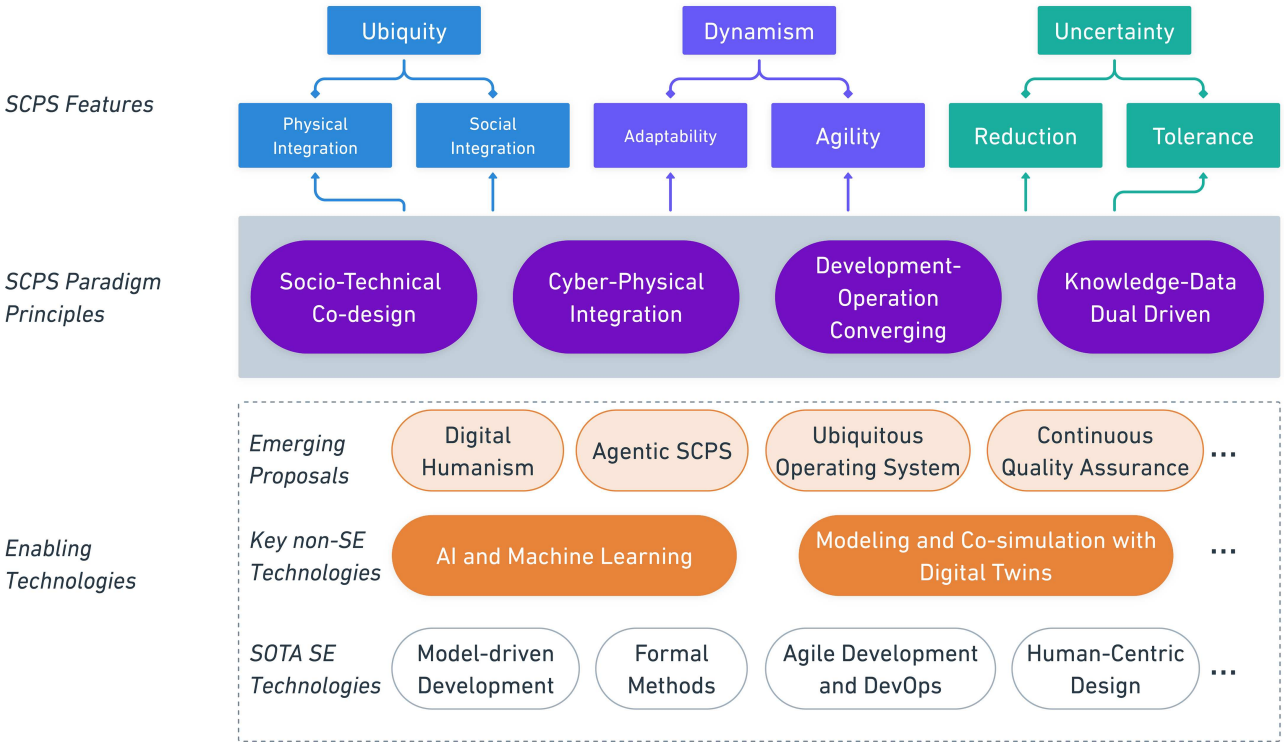


Figure 2 (Color online) SCPS features, principles, and technologies.

2.3.2 *Cyber-physical integration into the Mirrorworld*

The underlying infrastructure of SCPSs, as best described by futurist Kevin Kelly's [31] vision of *Mirrorworld* (although David Gelernter first popularized the term), seamlessly merges the physical world with cyberspace. This synergy is profound because of the unprecedented *ubiquity* and *depth* of digitalization. In terms of ubiquity, every physical object will be faithfully represented by a *digital twin* in cyberspace, through which the state of the physical object can be observed, and actions upon the digital twin are reified into the physical object, all in real time. In terms of depth, the digital twin provides not only a natural interface for users to interact with (cf. *augmented reality*), but also a faithful model of the context and behavior of the object, with which simulations and predictions can be made in cyberspace at runtime. The model can be manually developed, but the trend is to learn it from the data collected.

Moving into the Mirrorworld will significantly enhance the capabilities of computing-based solutions for real-world problems. Methodologically, Mirrorworld offers a universal abstraction for SCPSs. It fully supports the idea of "*instead of building computing into the device, the device must be built around computing*" [1]. Technically, it greatly broadens the scope of accessible resources and uplifts the level of business intelligence implementation. For example, intelligent transportation requires not only automated driving vehicles but also a comprehensive set of networked digitalized facilities on roads and in the computing cloud, and smart manufacturing involves not only robots but also the digitalization of the 5M1E, namely Man, Machine, Material, Method, Measurement, and Environment (see Subsection 5.2 for detailed discussions).

2.3.3 *Development-operation convergence for agility and adaptability*

In open-ended and dynamically evolving domains, many SCPSs will never be truly finished, as they must continually adapt to ongoing changes in their environments and requirements. Therefore, the engineering of such systems should be *adaptable* in the sense that the changes are anticipated as much as possible, and the system is designed with the flexibility to accommodate the changes. Moreover, it should be *agile* in the sense that unforeseen changes can be swiftly handled through effective redevelopment and redeployment.

The agility and adaptability are achieved through the convergence of development and operation of SCPSs. On the one hand, some development capabilities are "internalized" into the SCPSs, enabling runtime adaptations by the SCPSs themselves. For example, self-adaptive systems built with the MAPE-K model contain elements that *monitor* changes in the system and its environment, *analyse* the need for adaptation, *plan* for adaptation and

execute the adaptation plan, and the element of shared *knowledge* [32]. On the other hand, an agile approach for SCPSs will integrate development and operation processes to support rapid development and continuous delivery of system updates, driven by ongoing and timely feedback from operation. This agility and adaptability, along with the increasing level of automation in development based on AI and the ability to update a system without stopping it [33], allow SCPSs to grow, meaning they experience continuous evolution and a long-lasting lifespan despite changes in their environment and requirements.

2.3.4 *Knowledge-data-driven approach to taming uncertainty*

To address the vast and various uncertainties faced by SCPSs, multiple means at different levels of granularity are needed. Some of these means are explicit rules derived from human knowledge, and others are implicitly encoded in neural network models trained on data.

- *At the implementation level*, software elements in different forms of existence and construction methods are involved. In his keynote speech at AI Startup School, Andrej Karpathy summarized three types of software today: program code, neural network weights, and LLM prompts. He called them Software 1.0, 2.0, and 3.0, respectively [34]. While Software 1.0 handles uncertainty through encoded human knowledge, Software 2.0 addresses it through probabilistic approximation, and Software 3.0 helps address the frame problem through a form of common sense.
- *At the architecture level*, support for context-awareness and self-adaptation [32] should be included to ensure resilience against uncertainty. The monitor-analyse-plan-execute loop of self-adaptation can be viewed as a behavioristic approach to taming uncertainty.
- *At the process level*, the agility of software updating, which is discussed above, enables recognizing the uncertainty faced by SCPS during operation and addresses it through rapid system evolution. A model-driven and AI-enabled development-operation convergence process is uncertainty-resilient.

2.4 State-of-the-art of software development methods and enabling technologies

Next, we revisit state-of-the-art software methodologies for complex systems to discuss which aspects apply to SCPSs and which require adaptation due to new requirements. We list some of the existing methods that will evolve for SCPSs and select specific enabling technologies to support the paradigm shift resulting from the incorporation of human and social elements.

2.4.1 *State-of-the-art methods*

Test-driven requirements engineering. SCPSs demand a robust and responsive engineering approach that accounts not only for technical performance but also for social interaction, user behavior, and systemic complexity. In this context, requirements engineering (RE) and testing must converge early to manage interdependencies and mitigate risks. Test-driven requirements engineering (TDRE) [35] offers a practical and scalable solution: test cases are developed in parallel with requirements, enabling immediate validation and feedback on feasibility, consistency, and testability. This parallelism replaces traditional sequential models, where testing occurs too late to influence specifications. In TDRE, each requirement is augmented with structured, traceable test cases that follow the same formalism, aligning with the principles of test-driven development (TDD) but lifted to the specification level. This methodology proves particularly valuable in SCPS, where human-centered functions and real-time responsiveness introduce high variability and safety concerns. Regression tests are attributed early to enable future automation, while effort estimation and risk identification become more precise from the outset. By embedding test logic into requirements, TDRE fosters transparency, improves stakeholder communication, and accelerates compliance with safety and quality standards—an essential advantage for developing dependable socio-technical systems in mobility, healthcare, energy, and smart infrastructure.

Model-driven engineering. Model-driven or model-based software engineering has played an important role in the past decades [36,37]. It uses (semi-) formal, machine-readable and processable models to manage the complexity by abstraction and automation support. Towards SCPS, models will be even more important for supporting the interdisciplinary domain and ensuring consistency and traceability across heterogeneous components of the human, cyber, and physical domains. On the other hand, model-driven engineering has a solid foundation for automating transitions from design to implementation and producing software systems, and these transitions will be augmented by newly enabling technologies such as AI [38]. As an abstraction of SCPS, the modeling of SCPS should provide a unifying layer that abstracts operational details and presents programmable interfaces for managing heterogeneous

resources effectively. Specifically, learning-enabled components make the kinds of models even more diverse. Model-driven methods should embrace the uncertainties from machine learning models such as perception models and LLMs in the system. Furthermore, such a software-defined approach should enable system autonomy, which incorporates knowledge-and-data dual-driven self-optimizing and self-healing capabilities to make SCPS adapt intelligently to dynamic environments.

Agile development and DevOps. Agile development has been widely applied in current software engineering. It has the features of rapid iteration and collaboration. It has been frequently used in the development of internet-oriented software with humans in the loop. Agile development enables frequent system delivery in response to volatile and fast-changing requirements [39, 40]. When approaching SCPS, it must consider the physical components and the need to integrate with rigorous software engineering practices. The underlying software architecture will be important to support compositionality and divide-and-conquer principles with agility over heterogeneous components. Such system-level agility also requires agility of organizational structure to fit the socio-technical nature of SCPS [41]. This will raise the security and efficiency requirements of SCPS, since malfunctioning or processing delays may result in severe consequences. DevOps enhances system security and sustainability through adaptation and evolution in the loop. DevOps automates software deployment and testing, as well as continuous monitoring, data collection, and self-adaptation. It is notable that DevOps makes system development and evaluation a living process in the sense that the system can learn, adapt, and evolve under its open environment to deal with uncertainties. Such a paradigm may effectively promote system resilience. Towards SCPS, it should be capable of integrating systems from the information technology and operation technology domains. Similar to the relation between cyber components and physical components in cyber-physical systems, there is a gap that IT systems often concern security, interoperability, and continuous maintenance with frequent updates, while OT systems often run with a longer lifetime and are often with high integrity, availability, and safety. DevOps must be extended to bridge this gap through enriched software-defined capabilities. Furthermore, DevOps is essential for the evolution of SCPS, which requires ongoing refinement of system software design and continuous learning within the learning components during system operation. It is necessary to develop more sophisticated decision-making algorithms that incorporate real-time learning and predictive analytics. Agile development and DevOps will be combined in SCPS development with enabling technologies, in particular digital twins, to achieve software agility while maintaining the safety and reliability required for physical and critical infrastructure. They are intended to enable SCPS to continuously evolve, thereby making the system inherently more resilient and better aligned with real-world human needs.

Formal methods. Formal methods are mathematically grounded, rigorous methods to specify and verify computer and software systems [42, 43]. With the wide recognition of computational thinking, formal methods have become an underlying foundation of abstraction and automation. This favors at least two aspects. On the one hand, the introduction of autonomy and interconnectivity in SCPS increases the potential for cascading failures, in which a single erroneous decision or communication failure can propagate across a network, leading to large-scale hazards. Formal methods serve to guarantee the correctness, safety, and security of systems, and provide "provable guarantees". Integrating safety-critical constraints into the design process ensures we detect failure scenarios before deployment. On the other hand, it supports computer manipulation of the artifact and make development and checking automated. It is well known that formal methods will become increasingly important for developing correct systems. The benefits include minimizing exploitable software vulnerabilities and increasing reliability and robustness. Formal methods can promote the unification of techniques from different discipline domains to realize interdisciplinary integration. Developing a formal foundation that integrates diverse modeling techniques can improve interoperability and ensure verification results extend across the system. Towards SCPS, formal methods face a grand challenge: scalability. Compositionality should be increasingly important for applying formal methods to large systems. Secondly, formal methods should be seamlessly integrated into the pipeline of SCPS development and deployment, that is, in a continuous manner used by developers. This needs careful consideration to integrate formal verification at the key parts and phases with the agile/DevOps methods. For example, we may apply formal methods to constrain simulation and testing. By deriving formal properties that the system must satisfy, we can guide simulation-based testing towards critical corner cases. In addition, formal methods will face a challenge brought by machine learning (ML) components in the systems [44]. At the moment, ML components are often black boxes in the system; this not only makes formal verification necessary but also makes it difficult. We need to deploy lightweight formal monitors in operational vehicles to enable continuous assurance of safety and security properties.

Human-centric design and data engineering. Since humans are integrated into the SCPS, people's requirements and experiences are the priority in system development. Human-centric design (HCD) aims to optimize human-machine interaction and make the system more effective and safer. It increases system usability, decreases

cognitive overhead, and improves operator performance. It includes identifying and validating user roles, collecting and analyzing data, and clarifying user needs. Recently, HCD has been extended to integrate AI to accelerate design iterations and enhance the user experience. Towards SCPS, HCD will be expected to address ethical issues such as bias, discrimination, and potential negative societal impacts. It plays a key role in shifting the design paradigm from a single technical view to a social-technical view. In SCPS, HCD will address human-AI collaboration beyond human-machine iteration [45, 46]. Data streaming and collection have been increasingly extensive in SCPS. Not only data and data flow, but also data management, such as privacy and security, as well as other value-related properties, must be systematically considered. It means that SCPS development must incorporate value-preserving data engineering beyond traditional software engineering. Therefore, it also makes SCPS a technical-social system.

2.4.2 *Enabling technologies from other disciplines*

AI and machine learning. Software engineering has been reshaped by AI, particularly LLMs [47]. For example, machine learning techniques can generate functional code according to the requirements prompts and adversarial test cases to explore the system's behavior under extreme conditions. We notice that multimodal foundation models will have great potential for the development of SCPS, because data types are essential across multiple modalities [48]. On the other hand, what could be more challenging is that AI and machine learning components will become important parts of SCPS and perform perception, decision, and action functionalities [44]. Such components will perform the uncertain connections among human, cyber, and physical components. For example, as data intelligence engines, such machine learning models will serve in perception, planning, and decision-making roles. Although foundation models play as the most enabling technology for the development of SCPS, we note that their hallucinations and uncertainties introduce new risks to the correctness of SCPS with learning-enabled components and subsystems. AI and machine learning require us to address issues not only in human-machine collaboration in software development, but also in neuro-symbolic forms of SCPS. What should be noted is that with AI, the boundary between system development and system operation will converge. Autonomy and adaptation will be the system development goal as well as the inherent system behavioral goal, in the sense that "development" will be "internalized" in the SCPS. AI will move forward as human-centered, autonomous, embodied agents that can perceive, reason about, and change the physical world, and will interact and collaborate with humans. An ethical AI development and governance framework will also be part of the value stream within data engineering for SCPS.

Modeling and co-simulation with digital twins. Modeling is the most challenging aspect of SCPS development. This is because SCPS models are often not clearly specified, for example, in modeling the hybridity of discrete and continuous systems, human intention or behavior, and data-driven machine learning components. Therefore, modeling and simulation over models are becoming challenging in the development, monitoring, and operation of SCPS. Fortunately, digital twin technology provides an enabling approach to addressing such challenges [49, 50]. Digital twins (DT), originally used to build virtual replicas of physical systems, will be extended to model components of humans, cyber, and physical systems. It will employ a model-driven engineering approach to enhance the validation and operation of SCPS. DT can enable continuous monitoring, failure prediction, proactive maintenance, and rapid validation of systems. It can also be used as feedback to adapt the design model for optimization and evolution. As we mentioned before, it will greatly enhance the agility and adaptability of SCPS. Towards SCPS, DT will serve as a living digital mirror of the system, making model-driven, agile, and DevOps feasible to support self-healing and continuous adaptation in SCPS. For example, DT will support continuous experimentation to validate the changes in the real operational environment before deployment. Therefore, it is essential to ensure the quality, integrity, and real-time synchronization of data moving between the physical and social worlds and the digital mirror. The interplay between digital twins and physical entities will require more seamless integration techniques. For example, unified modeling across heterogeneous disciplines is critical to enhance the fidelity and responsiveness of SCPS. Finally, as human factors become increasingly central, system software should evolve to support intuitive human-in/on-the-loop interactions, ensuring that the system aligns with user needs and societal values.

2.5 Emerging proposals

Beyond the developments mentioned above, the push for a new paradigm has inspired several novel proposals. Rather than attempting a complete survey, we discuss four representative proposals—digital humanism, agentic SCPS, ubiquitous operating systems, and continuous quality assurance—that together illustrate complementary aspects of the emerging paradigm: its socio-technical values, development methodology, runtime support, and quality assurance, respectively.

2.5.1 *Digital humanism*

Software is the defining technology of the 21st century, shaping how humans interact with the physical environment, cooperate with each other, and organize society at all levels—political, industrial, educational, and social. Most of this software is human and societally critical (HSC), directly impacting individuals and society. However, the current state-of-the-art in software engineering (SE) research and practice is inadequate to address the challenges posed by HSC systems, often overlooking their potential threats to privacy, fairness, and societal well-being [51]. Advances in AI have further amplified these challenges, exposing the limitations of traditional SE paradigms.

A paradigm shift in SE research is needed, placing HSC concerns at the forefront of software design, use, and evolution. SE needs to develop new theories, methods, and tools to systematically address these concerns. These are examples of crucial questions that need to be answered.

- How can HSC requirements, such as fairness, privacy, sustainability, and security, be systematically identified, classified, and specified in actionable ways?
- How can awareness and responsibility among all stakeholders (designers, regulators, users, etc.) be improved to ensure HSC concerns are effectively addressed?

The societal nature of software has long been recognized by social and political scientists. In the late 1990s, Lawrence Lessig observed that software (code) acts as the regulator of the digital world [52]. He noted: this code, or architecture, sets the terms on which life in cyberspace is experienced. It determines how easy it is to protect privacy, or how easy it is to censor speech. It determines whether access to information is general or whether information is zoned. It affects who sees what, or what is monitored. In a host of ways that one cannot begin to see unless one begins to understand the nature of this code, the code of cyberspace regulates.

SE research needs to address these challenges by redefining traditional practices, ensuring that software systems align with ethical, sustainable, and socially responsible principles [53]. To align SE with HSC goals, we first need to identify the set of fundamental conceptual values and principles upon which it has to be founded: those of digital humanism [51]. Humans are at the center of our concerns; technology must serve humans and respect their values rather than the other way around. We need to bridge technical development with human-centered values to create a foundation for responsible SE in the digital age.

The consequences of neglecting HSC concerns in technological development can be severe, especially as research is rapidly transferred into practice without sufficient validation. This is particularly true for AI, which enables the creation of larger, more complex systems that may exhibit unforeseen behavior. Complete control over such systems is unattainable, and the illusion of control can be dangerous [51, 54].

Despite the profound impact of software on humans and society, it is still largely treated as a purely technical artifact. Likewise, SE is still considered to be a purely technical discipline that is based on purely technical theory and methods. SE research has traditionally focused on supporting the development, deployment, and evolution of quality software within controlled production time and budget. While technical qualities such as correctness, efficiency, and usability are verified and validated, HSC goals and impact are largely ignored, and no systematic methods have been investigated to address them. Often, ignorance is an unconscious choice: they are unknown unknowns. Unanticipated uses and their potential effects are also ignored during development. To support the responsible design of HSC systems, we need to raise awareness about HSC concerns and focus development on their explicit elicitation and guiding design to address them through explicitly documented and responsibly made decisions. In the case of HSC systems, this process requires multi-disciplinary contributions, enriching technological competences with others. Depending on the specific system, they may range from humanities (notably, ethics), social sciences, psychology, political science, health, law, ecology, etc.

A typical crucial decision that often needs to be made is the extent of automation or even autonomy a system should exhibit. The Germanwings example, where a pilot could take control over the automatic system with catastrophic consequences, shows that the simplistic notion that "humans must always retain ultimate control over autonomous system actions" can be equally bad as the one that fully relies on complete system autonomy. Deciding the level of autonomy is a critical design decision, which should be the result of an ethical deliberation process, and decisions should be made responsibly and explicitly specified.

We argue that an initial roadmap to address HSC concerns should include the following steps.

(1) Develop a conceptual model and precise, formal definitions for HSC requirements, including their dependencies and trade-offs with traditional technical requirements.

(2) Align existing verification techniques, including runtime and continuous verification, with HSC requirements to ensure their satisfaction throughout the software lifecycle.

(3) Define a multidisciplinary software development process that integrates contributions from other relevant fields, like social sciences and ethics, to address HSC concerns.

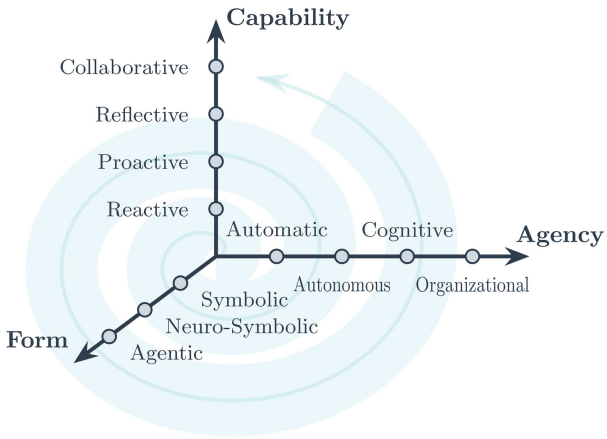


Figure 3 (Color online) Development toward agentic SCPSs.

(4) Conduct iterative assessments of the proposed theories, methods, and processes.

(5) Innovate SE education vis-a-vis progress in previously mentioned areas. Academia provides the ideal setting in which students can be engaged in multidisciplinary learning processes. Students and researchers from different disciplines can work together and learn to speak a common language, through which they can gain a deep understanding of the mutual relations among digital technology, humans, and society. Students can then be testimonials of a new approach to designing HSC systems according to DigHum and bring innovation to the industry.

2.5.2 *Agentic SCPS*

To address the aforementioned challenges of SCPSs, an emerging approach is to build the system as a team of intelligent agents. This agentic approach shifts away from specifying the concrete behavior of each element in the system toward constructing a goal-oriented collaboration of agents. The agents make their own decisions, take actions, and collaborate with each other to pursue their individual and collective goals. The benefits are attractive: agents can adjust their behavior at runtime based on perceived context and learned experience, rather than being limited to rigidly predefined rules.

For example, consider an emergency rescue scenario in a smart city where a severely injured person requires immediate transport to a hospital. In a conventional cyber-physical design, traffic lights might be rigidly programmed to respond to the ambulance's proximity sensor, attempting to clear intersections sequentially. However, this reactive approach fails if adverse traffic conditions or temporary construction block the route further ahead. In an agentic solution, the ambulance agent broadcasts a high-level goal: "*I have an urgent situation and need to minimize transit time to the hospital.*" It consults the city traffic agent and negotiates with distributed traffic-light agents to not only clear a path but to dynamically select the optimal route based on real-time road conditions. Even further, the agents may collaboratively decide to retarget a different hospital and notify the hospital agent to prepare a surgical team, prioritizing patient survival over the original route.

While this concept is intuitively appealing, implementing it with conventional software technology turns out to be very challenging. The difficulty stems from the inherent uncertainty of high-level goals—for instance, what "urgent" means depends on different contexts—and the impossibility of predicting every runtime scenario in an open environment. Encouragingly, the feasibility of this approach is increasing with the rapid development of AI technology, particularly LLMs. LLMs excel at handling the ambiguity of such tasks and interpreting rich human intentions expressed in natural language.

Currently, the interest in agentic AI is tremendous and growing fast [55–59]. However, the software engineering for agentic systems is still in its early stages. Figure 3 illustrates a developmental viewpoint toward agentic SCPS.

Capability. Agents in an SCPS transcend simple reactivity; they must be proactive, reflective, and collaborative. This implies they are not only capable of perceiving their environment, understanding the situation, making decisions, and taking actions based on predefined rules and models (cf. OODA and MAPE-K architectures [32]), but also possess the "meta-level" ability of self-awareness, self-improvement, and goal-setting based on their internal intentions. Furthermore, they do not merely reflect on themselves but also reason about others to collaborate toward common goals. These capabilities require structural facilities organized according to some reference architecture, e.g., the one described by Sifakis et al. [59].

Agency. Agency is the degree to which an agent can achieve high-level goals in open environments with limited

human direction. While traditional software is designed to be automatic, agents are autonomous in the sense that they make decisions on their own, and cognitive in the sense that they reason at the *semantic* level. For example, the ambulance agent mentioned above understands the semantic implication that its purpose is "to save a life" and that "time to hospital" is a critical constraint. Building upon this semantic-level reasoning, agents can further ascend to an organizational level by teaming up with other agents or humans to complete complex, collective tasks.

Form. The implementation form of agentic software is inherently hybrid. It consists of (1) conventional symbolic program code that is efficient and reliable but limited to fully determined computations; (2) artificial neural networks suitable for data-driven tasks of classification, regression, and generation; (3) natural language prompts that direct underlying LLMs to express the semantic-level direction for agency.

Despite the impressive capabilities and rapid progress of agentic AI, significant hurdles remain for the engineering of agentic SCPSs. While challenges such as security risks [58], governance [55], and physical embodiment [60] are well-recognized within the broader AI community, other technical difficulties persist for software engineers and are as follows.

Semantic alignment. Constructing a coherent system from hybrid components requires bridging the semantic gap between paradigms. The central challenge lies in automatically aligning concepts among the logical semantics of symbolic code, the high-dimensional representations of neural networks, and the expressions of natural language prompts. Ensuring consistency across these three modalities is critical for system robustness.

Control-autonomy balance. Adopting an agentic approach necessitates embracing the inherent uncertainty of autonomous behavior. However, disciplined software engineering must carefully balance this autonomy with control. The goal is to provide sufficient predictability and reliability guarantees through cost-effective mechanisms, preventing the system from drifting into unsafe states while still allowing the flexibility required for agency. For example, there has been an increasing interest in runtime enforcement for agents [61, 62].

Scalability and evolvability. The scalability of agentic systems remains an open question, primarily due to the lack of principled compositionality in current AI agents—unlike traditional software modules, agents do not easily compose into predictable larger systems. Furthermore, lifecycle management is complex: it is unclear how to smoothly and incrementally co-evolve an agent system alongside its underlying foundation model (e.g., LLM), which changes independently of the application logic.

2.5.3 *Ubiquitous operating system*

SCPSs represent a new form where the boundaries among human society, cyberspace, and the physical world are blurred through ubiquitous and seamless connectivity. To support diverse SCPS applications, there is an urgent need for a new class of operating systems (OS). We can consider the role of an OS for SCPSs from the resource and application layers, as it essentially acts as a mediator between the two.

At the resource layer, SCPSs run in the ubiquitous computing environment that deviates significantly from traditional cluster-level or data center-level computing, to enormous Internet of Things (IoT)/edge devices, including smartphones, autonomous vehicles, robots, and so on. As a result, the resource types vary significantly, and their operation conditions change frequently.

At the application layer, with the recent breakthroughs and wide adoption of artificial intelligence, i.e., machine/deep learning techniques and the recently popular LLMs, AI modules have become key components in modern software applications. In particular, the emergence of agents marks a shift in how a task is invoked, orchestrated, and experienced, with profound implications for OS design. Unlike traditional applications, agents are goal-driven, continuously executing, and context-aware, blurring the boundary between user intent, system services, and application logic. The rise of AI-powered applications and agents suggests a shift in system design from managing static programs to managing adaptive, learning-enabled entities. Operating systems and runtime must increasingly treat AI components and agent state as schedulable, governable objects.

Key design principle: software-defined. Given the paradigm shift of SCPSs as well as the role of operating systems, we consider that the OS is no longer a static substrate with fixed, predefined functionalities. Moreover, an OS running in the ubiquitous computing environments must contend with profound heterogeneity across scenarios, including variations in resource scale, functionality, performance requirements, security guarantees, and ecosystem constraints. As a result, it is impossible to build a "***one-size-fits-all***" general-purpose operating system. Instead, we argue that a new class of OS is urgently required. Crucially, these OSs should not be envisioned as a single, monolithic, general-purpose OS. The OS should follow the basic design principle of "*software-defined*". Technically, software-defined refers to resource virtualization and function programmability. In practice, software-defined mechanisms encompass a range of solutions, including software-defined networking (SDN), software-defined storage (SDS), and software-defined data centers (SDDC). Collectively, these are often called "software-defined everything"

(SDX). In these systems, hardware resources are virtualized, allowing them to be managed by a control plane or operating system routines. This "software-defined" principle enables user-defined programs to access and manage the services provided by these virtualized resources. From this perspective, OSs and software-defined systems are mostly based on the same principles.

The UOS proposal. Following the software-defined principle, the OS for SCPSs can be conceived as a class of domain-specific and scenario-aware systems, each tailored to the demands of its application context. Beyond a lightweight, general-purpose, and customizable "core", the OS should become a highly software-defined system whose form and behavior are flexibly instantiated by the requirements of specific application domains and contextual settings. A notable proposal of a new design space for OS is the idea of "***ubiquitous operating systems*** (UOS)" [63]. UOS suggests that the future of OS design must embrace a software-defined principle. UOS constitutes a new type of OS for a software-defined world where software will be used to manage all aspects of cyberspace, the physical world, and human society. In a narrow sense, a UOS is mainly designed toward the ubiquitous computing environment with a huge volume of devices, focusing on lightweight computation, real-time responsiveness, and reliability, while providing common support for the core functions of sensing, communication, cognition, and control. In a broader sense, a UOS can also be viewed as a new form of middleware-layer system software designed for networked environments and contextualized scenarios. Built atop existing node-level operating systems, it integrates and coordinates device capabilities across platforms, serving diverse application scenarios such as IoT/edge, robotics, smart cities, smart manufacturing, and smart homes. Core system software developed for different domains can thus be regarded as concrete instantiations of the ubiquitous operating system paradigm along specific dimensions.

Intuitively, UOS supports advanced capabilities in the form of pervasive sensing, pervasive connectivity, lightweight computation, lightweight cognition, feedback-driven control, and intelligent user interaction. Technically, the UOS should embody the same key concepts as traditional operating systems, including resource virtualization and function programmability. However, these concepts are more specifically defined for SCPS scenarios.

Resource management. UOS should provide system-level abstractions for managing a diverse set of resources that extend beyond compute, storage, and network, such as physical objects, humans, and so on. Analogous to drivers or hardware abstraction layers in traditional operating systems, these abstractions facilitate resource virtualization in a more general and unified manner. Furthermore, UOS should provide well-defined APIs through which users and applications can interact with and utilize these virtualized resources. For example, in the context of social networking, UOS should manage resources encompassing user identities, social relationships, behavioral activities, and communication exchange.

Development and runtime. UOS should furnish comprehensive development and runtime support for applications. Similar to traditional operating systems, it provides APIs, programming models, libraries, and associated development tools. In particular, the AI module becomes a key component of modern applications. As a result, in addition to the platform support for business logic development, UOS should provide lifecycle support for data processing, model training, fine-tuning, and model inference. In addition, running agents requires the underlying software stack to manage model execution, state persistence, tool invocation, and coordination across multiple agents and resources, and thus introduces long-lived, adaptive workloads whose behavior depends on both learned representations and explicit program logic. These supports are situated at a higher level, where AI applications execute atop the existing OS, which itself operates above conventional operating systems such as Linux or Windows. In practice, we argue that there are three key principles for UOS.

• **First, UOSs are inherently scalable across a wide spectrum of system sizes and deployment environments.** Traditional operating systems have successfully evolved from resource-constrained embedded platforms and mobile devices to desktop and workstation systems, and further to large-scale server clusters and cloud infrastructures. We envision UOSs extending this trajectory to encompass both legacy and next-generation computing platforms, ranging from ultra-low-power edge devices to globally distributed computing fabrics spanning multiple administrative domains. Beyond conventional workloads, UOSs are expected to natively support emerging application domains such as big data analytics and artificial intelligence.

• **Second, a UOS can serve as the system software substrate for individual physical objects or collections of objects in the physical world.** A central objective of ubiquitous computing is to push computation beyond traditional IT systems and embed intelligence into everyday artifacts. Achieving this vision requires making such entities programmable, which in turn necessitates operating system-level abstractions for resource management, execution, and coordination. This trend is already evident in robotic systems, where even low-cost platforms are managed by specialized OSs. In smart environments, household appliances—including displays, appliances, lighting, and consumer devices—must expose programmable interfaces, while mobile entities such as vehicles, drones, bicycles, assistive devices, and other cyber-physical systems require a UOS to manage sensing, actuation,

communication, and computation.

• **Last but not least, UOSs are equally applicable to entities in the virtual world.** Beyond physical systems, operating systems can be instantiated for domain-specific virtual entities, such as organizations and social structures. Families, enterprises, institutions, and government agencies can be viewed as software-defined systems that manage heterogeneous resources including personnel, information, schedules, and inventories. In this context, a UOS provides uniform abstractions for resource management and coordination, as well as a runtime substrate for developing and executing applications that operate over these virtualized entities.

2.5.4 *Continuous quality assurance*

Like traditional ICT systems, the quality assurance of SCPSs involves modeling, specification, and assurance techniques—primarily validation, testing, simulation, verification, synthesis, monitoring, etc. As discussed above, additional distinguishing features of SCPSs, such as self-evolution and high uncertainty, do introduce fundamental changes to the quality assurance process.

• From the dimension of properties. Beyond technical properties common in traditional ICT systems (e.g., safety, liveness, QoS, security, and privacy), we must consider non-technical properties such as human intentions, values, and social-collaborative objectives, etc.

• From the dimension of assurance processes. It is necessary to shift from the posterior guarantees of traditional ICT systems to the posterior plus ex-ante guarantees required for AI-enabled CPS, and finally to the incremental and self-evolving guarantees for SCPS across co-design, co-production, and co-delivery.

• From the dimension of assurance techniques. We have to follow the roadmap evolving from symbolic computation-based techniques for traditional ICT systems to neuro-symbolic techniques for AI-enabled CPS, and ultimately to neuro-symbolic-agentic computation-based techniques for SCPSs. Specifically, different techniques must be integrated uniformly to resolve diverse assurance goals: validation for requirement rationality and completeness, simulation and testing for prediction and adaptability, verification/monitoring for safety-critical properties, synthesis for achieving collaborative objectives, and so on. Particularly, all the aforementioned techniques should be combined with learning-based techniques, as advocated by some first attempts [64–67].

To address complexity and scalability in SCPSs quality assurance, our principle is "divide and conquer", fully exploiting model-based and data-driven design complemented by design-by-contract. In what follows, we sketch a potential solution towards the changes in the three dimensions discussed above, in which the notion of digital twin (DT) plays a central role.

A digital twin for an SCPS is a comprehensive framework comprising models and quality guarantee services aimed at facilitating the design, analysis, adaptation, and evolution of complex systems. It integrates models representing social, organizational, and technical aspects. This digital twin enables stakeholders to gain insights into system dynamics, evaluate scenarios, and make informed decisions to optimize performance and resilience in dynamic environments.

Building an initial digital twin to support self-evolution. The envisioned digital twin must effectively manage the multi-level and multi-dimensional complexity inherent in SCPS. To achieve this, holistic modeling approaches are indispensable, complemented by predictive and adaptive processes. This approach enables us to comprehend intricate interactions among various models, ensuring they accurately represent real system behavior, which will play a central role in the model-based design for complex systems. Furthermore, it facilitates the identification of anomalies and inconsistencies, allowing for timely model revisions to maintain alignment with evolving real-world dynamics.

To implement this, we suggest applying data-driven methods alongside domain knowledge to first construct digital models of social components. Then, by exploiting the separation of concerns, we can ensemble existing modeling techniques from different fields to obtain a multi-layer, multi-view, and multi-dimensional model. The key issue here is how to guarantee compositionality. A simple solution could be applying the orthogonal product technique, like the integration of CSP, OZ, and DC [68]. Possibly, we need design by contract to define a set of rules and constraints on how to ensemble and interact these highly heterogeneous sub-models. Additionally, as SCPS components are highly interconnected, changes in one area often trigger a ripple effect. We must, therefore, establish evolution methods capable of handling these changes within the digital twin in a highly automated manner.

Refining the digital twin with real-world data. The SCPS digital twin must be connected to data from the software system and the physical environment. On the one hand, the model can be validated and refined via this data; on the other hand, the real system can be monitored and enforced by comparing the observed behaviors against the predicted behaviors derived from simulation or reachability analysis of its digital twins. To this end, we must: (1) develop approaches for data sensing and fusion to collect real-time, multi-modal data; (2) establish

a mapping mechanism to link this data to specific sub-models, typically subject to the set of rules and constraints defined in a contract.

Adaptive simulation and reachable set computation for prediction and monitoring. Simulation is a vital tool for stakeholders to gain insights into system dynamics and evaluate scenarios. However, simulating an entire SCPS is challenging due to inherent complexity, heterogeneity, and uncertainty, whose bottleneck is scalability. A potential solution is to conduct simulations over loosely assembled sub-models and then exploit their interconnections according to the contract to derive a total simulation result.

Furthermore, to guarantee safety in safety-critical CPS, simulation must be complemented by reachable set (or reach-avoid set) computation. While simulation is inherently incomplete, reachability analysis provides a formal trade-off between safety guarantees and computational efficiency. Consequently, we need to extend existing CPS reachability analysis techniques to the specific requirements of SCPS digital twins, particularly for societal subsystems.

Incremental and regressive verification and monitoring. Verification and monitoring of safety-critical SCPS are mandatory but fraught with challenges (see Section 4). To maintain efficiency, we utilize the principle of compositionality inherent in the digital twin. Theoretically, this requires new theories and algorithms for reasoning within neuro-symbolic-agentic computation. LLM-based techniques combined with numeric-symbolic optimization, specifically the combination of SMT and OMT, offer a promising path. The seamless integration and management of diverse techniques (learning, validation, testing, simulation, verification, and synthesis) is crucial to provide an incremental, self-evolving guarantee mechanism, as showcased in [64–66].

Synthesizing strategies to drive societal change. An SCPS digital twin can be viewed as a multi-agent system consisting of autonomous agents with varying stances toward technology, differing competencies, and distinct sub-objectives. These agents must cooperate with each other to achieve social-collaborative objectives. A crucial challenge lies in decomposing these high-level objectives for players with diverse profiles and cultural backgrounds. This necessitates the development of new game theories for multi-agent systems that account for heterogeneous objectives and constraints.

3 Challenges of ubiquitous operating systems for SCPS

In the computer stack, an OS is system software that manages computer hardware and software resources, and provides common services for computer programs. In the history of modern computing, the OS has always played a crucial role. As mentioned above, SCPS implies complex ecosystems that blend computational intelligence, physical interaction, and socio-contextual behavior. Therefore, in this section, we discuss the new requirements, challenges, and potential trends of building a new OS for SCPS that runs in the ubiquitous computing environment.

3.1 Emerging new requirements of operating systems

Given the characteristics of SCPS, operating systems for SCPS in the ubiquitous computing environment should meet the following requirements.

Supporting distributed and heterogeneous resource management. The OS uses vast, distributed, and heterogeneous resources spanning from the cloud data centers, edge devices, IoT devices, and even to physical objects and humans. OS can encompass compute, data storage, communication networks, and various software services. In addition to traditional resources like compute, storage, and network that are typically deployed on the cloud/server, more resource types that are dispersed on the IoT/edge devices across the ubiquitous environment, including sensors, actuators, physical objects, and even humans involved, should be abstracted and encapsulated as standardized, discoverable, and invocable services. As a result, the OS must provide deep virtualization and unified abstraction of resources. The wide spectrum of ubiquitous resources should enable a fundamental shift in resource management. This abstraction forms the foundation for cross-device resource pooling and lightweight, cooperative utilization.

Supporting intelligent and active interoperability. The OS must provide efficient interoperability to address extreme heterogeneity through effective bridging and coordinated orchestration. Faced with devices that differ widely in architecture, capability, and vendor, an operating system should establish general-purpose coordination protocols and intelligent scheduling mechanisms to dynamically and optimally assign tasks and data. Through such coordination, collections of simple devices can exhibit emergent behaviors that far exceed the capabilities of any individual node, enabling a paradigm shift from individual intelligence to collective intelligence. In particular, traditional OSs act primarily as passive arbiters of resources, allocating CPU, memory, and I/O in response to application requests. Especially, as agents require a more active orchestration fashion, the OS should continuously

mediate trade-offs across compute, memory, energy, latency, and intelligence quality. This is especially critical in heterogeneous environments, where agents must seamlessly coordinate CPUs, GPUs, NPUs, and domain-specific accelerators under dynamic workload conditions.

Supporting AI components lifecycle. Modern software applications increasingly embed AI as a first-class computational component rather than treating it as an external service. AI components operate continuously and interact tightly with traditional software logic, forming closed control loops that combine perception, reasoning, and actuation. AI agents extend this integration further by coupling AI components with persistent state, goals, and action interfaces. The rise of AI-powered applications and agents suggests a shift in system design from managing static programs to managing adaptive, learning-enabled entities. From a systems perspective, agents introduce long-lived, adaptive workloads whose behavior depends on both learned representations and explicit program logic. OS provides OS-level abstractions and runtime mechanisms that span the complete lifecycle of AI components, including distributed training, on-device or in-situ fine-tuning, and latency-sensitive inference. As model architectures grow in parameter count, memory hierarchy depth, and execution complexity, the operating system must jointly manage multiple tightly coupled constraints: heterogeneous compute availability across CPUs and XPUs, memory footprint and placement across device, host, and remote memory, energy and thermal budgets, and application-specific intelligence requirements such as accuracy, latency, and adaptability. Addressing these constraints requires coordinated scheduling, memory management, and power-aware control, enabling OS to make holistic, cross-layer decisions rather than isolated optimizations to promise the correctness, efficiency, and trustworthiness of intelligent behavior.

Supporting new user interaction model. The OS should support new human-computer interfaces beyond traditional user interfaces like keyboards and mice for desktops/laptops and touch-based interaction for smartphones. It yields a substantial improvement in more natural and intelligent user experiences. Future UIs should be moving beyond traditional graphical user interfaces (GUIs) and command-line interfaces (CLIs). Typically, future UIs can serve as intelligent and proactive assistants, automate tasks, integrate with agents, and so on. Hence, UIs are grounded in human-centric conversational modalities such as natural language, voice, video, gaze, body posture, spatial gestures, and potentially higher-level signals including attention and brain activity. These UIs aim to make interactions more intuitive, context-aware, and seamless. Additionally, rather than merely executing commands, the OS must support system-level agency—deciding when, how, and whether to act on the user's behalf. This raises foundational challenges in trust, explainability, governance, and user control, positioning the OS as the ultimate arbiter between autonomous behavior and human oversight.

Supporting long-term, continuous adaptation. The constant evolution of technology, including new hardware and APIs, forces developers to choose between costly system re-engineering or operating with suboptimal, outdated systems. Therefore, the OS needs to be intelligent and robust enough to adapt to changes in the physical and logical environments. This need for adaptability can originate from initiatives like DARPA's goal for software systems that can remain functional for over 100 years [69], outlasting their original designers and hardware. Moreover, new applications like agents exhibit long-running, adaptive lifecycles, maintaining state across time, context switches, and even device boundaries. In traditional OS, processes and threads are designed around relatively short-lived or well-scoped execution models. Supporting such demands requires persistent context management, checkpointing, migration, and cross-device continuity, while preserving isolation and fault tolerance.

3.2 Research challenges and opportunities for UOS

The preceding requirements present significant challenges that extend beyond traditional operating system design principles. As mentioned, UOS is motivated as an OS foundation for programmable, intelligent, and interconnected objects. In other words, UOS does not refer to a specific OS like Windows, Linux, or Android. Instead, it provides an abstraction of operating systems that can manage diverse and heterogeneous resources spanning cyberspace, the physical world, and human society. At the technical level, there are some, but not limited to, the following challenges to building UOS in practice.

Challenges of OS abstractions. A single and general model or architecture is unlikely to be sufficient for the diverse spectrum of UOS deployments. A key architectural decision lies in the choice of abstraction and programming-interface granularity. Finer-grained abstractions increase generality and composability, but often introduce additional indirection and coordination overheads that can degrade application performance. Coarser-grained interfaces, in contrast, can improve efficiency but limit flexibility and reuse. Identifying and navigating this tradeoff between abstraction granularity and performance is a central challenge in the design of UOS architectures. Conventional OSs usually assume that computation is initiated through explicit application launches and mediated

by well-defined process and application boundaries. New AI-empowered applications like agents, by contrast, operate persistently and proactively, interpreting high-level user goals and dynamically decomposing them into sequences of system actions. This shift calls for OS abstractions that elevate intent to a first-class citizen, enabling the system to reason about what the user wants to achieve rather than which application should be invoked.

Challenges of monolithic architecture. Currently, most modern OSs are large, complex, and difficult to maintain. They often use a monolithic kernel that tightly couples all resource components, such as CPU and memory, assuming that they are local to a single machine. The monolithic architecture, like Linux, dominates in general-purpose scenarios such as servers and the cloud. However, in the SCPS context, there is an increasingly large application space of billions of IoT/edge devices like smartphones, robots, and vehicles, which require better performance and security. In addition, the monolithic architecture inherently lacks support for heterogeneous resources and makes it difficult to add, remove, or change components. Furthermore, integrating new, heterogeneous resources like ASICs or specialized sensors is a costly and painful process.

Challenges of robust and lightweight runtime. The resource constraints of IoT/edge devices preclude the use of heavy, cloud-native Docker containers. This necessitates a shift toward much lighter, faster process-level isolation mechanisms. As IoT/edge devices take on mission-critical roles (e.g., autonomous driving, industrial control), the security and reliability of OS kernels are paramount. Meanwhile, delays in resource provisioning can lead to suboptimal performance or, worse, system failures in safety-critical applications such as autonomous vehicles or smart healthcare. Timely resource allocation and provisioning are challenging given the scale of UOS. Besides new scheduling system designs, smarter resource demand prediction would be essential.

Challenges of resource virtualization. Virtualization is a foundational mechanism underlying modern operating systems and software-defined infrastructures. As UOSs push computation from centralized cloud platforms toward the network edge—including smartphones and resource-constrained IoT devices—traditional virtualization techniques become increasingly inadequate. A UOS performs deep virtualization of geographically and logically distributed computational units, sensing devices, and actuators, abstracting their capabilities into standardized, discoverable, and invocable services. It represents a fundamental shift from managing hardware resources to managing circulatable services, enabling resource pooling and coordinated utilization across devices and administrative boundaries. Moreover, UOSs require lightweight virtualization mechanisms that can efficiently abstract heterogeneous resources while minimizing overhead. Such mechanisms are essential to provide OS-level isolation, multiplexing, and programmability, and to enable software-defined IoT/edge systems across diverse deployment environments.

Challenges of programming models and languages. Existing general-purpose programming languages, such as C/C++ and Java, are primarily designed around the abstraction of a single computer system and its execution model. This assumption increasingly mismatches the requirements of UOSs, where applications span heterogeneous resources, entities, and administrative domains. To bridge this gap, domain-specific programming models and languages are needed to express domain semantics, policies, and coordination patterns more directly, enabling the efficient development of applications tailored to particular UOS instances, such as enterprise-scale operating systems.

Challenges of pervasive sensing integration. The OS must sense and react to environmental changes. The IoT/edge devices are increasingly equipped with a large number of heterogeneous, multimodal sensors, making efficient integration and precise synchronization a "first-order" systems concern. The fidelity of downstream perception and control pipelines depends significantly on the consistency and temporal alignment of sensor data. To address this, the OS requires a dedicated hardware module that aggregates data from more than ten sensors and enforces a shared time base, enabling accurate data acquisition and cross-sensor correlation.

Challenges of complex software stack. The OS needs to integrate a diverse array of functionalities, from environmental perception and engaging in physical interactions to executing complex tasks. This integration involves harmonizing various components such as sensor data analysis, sophisticated algorithmic processing, and precise control over actuators. Furthermore, to cater to the broad spectrum of robotic forms and their associated tasks, a versatile embodied AI software stack is essential. Achieving cohesive operation across these diverse elements within a singular software architecture introduces significant complexity, elevating the challenge of creating a seamless and efficient software ecosystem.

Challenges of scalability and heterogeneity. UOS operates large-scale and extensive spectrums of physical, cyber, and even social resources. To operate across devices with diverse architectures, capabilities, and vendors, a UOS establishes common coordination protocols and intelligent scheduling mechanisms that support dynamic and near-optimal placement of tasks and data. Through such orchestration, collections of simple devices can collaboratively exhibit complex behaviors that exceed the capabilities of any individual node, enabling a paradigm shift from isolated intelligence to collective intelligence. Particularly, the heterogeneity introduces various challenging

issues to AI models. While modern heterogeneous computing hardware provides essential computational support for intelligent workloads, contemporary OSs largely manage such hardware as specialized peripheral devices. Although AI models can rely on bespoke adaptations and hardware-specific designs to exploit particular accelerators, this leads to severe system fragmentation. This fragmentation not only complicates system design and maintenance but also substantially limits the scalability, portability, and widespread adoption of AI-empowered models like LLMs and agents.

Challenges of OS tuning for domain-specific purposes. From the behavioral perspective, traditional OSs are developed with a rather general-purpose design. Users typically install them with default configurations, which are no longer adequate for diverse and changing application needs. While operating systems offer various tuning options, adjusting these settings requires expert knowledge and can be a time-consuming and error-prone process.

Challenges of security and privacy. As control and coordination increasingly shift from hardware to software, UOSs become the primary control plane for entire systems and environments, and consequently a high-value target for adversaries. The expanded scope and programmability of UOSs amplify the attack surface, exposing vulnerabilities across heterogeneous components and administrative domains. Moreover, UOSs that manage sensitive personal data or mission-critical information must treat privacy as a first-class design objective, requiring principled mechanisms for isolation, access control, auditing, and policy enforcement across both physical and virtual entities.

3.3 Potential key enabling technologies

In this subsection, we briefly discuss some promising enabling technologies for building UOS.

3.3.1 *New OS abstraction design*

Unified abstractions for heterogeneous hardware. Currently, XPUs—referring to various accelerators, such as GPUs, NPUs, ASICs, and FPGAs—are extensively deployed in SCPS, from cloud to edge, to offload intensive computations from host CPUs. Rich application scenarios in SCPS pose diverse requirements for XPU task scheduling. XPUs have significant and evolving differences in hardware capabilities. For example, while most systems rely on the general-purpose programmability of GPUs for preemptive scheduling through code transformation, many mainstream NPUs and ASICs are non-programmable [70]. A unified abstraction is urgently required for supporting preemptive XPU scheduling. This abstraction should systematically encapsulate heterogeneity in hardware capabilities and software stacks, shielding higher layers from device-specific complexity while preserving generality. The abstraction should also expose a common scheduling interface across diverse XPUs, enabling hardware-agnostic scheduling policies and coordinated execution among heterogeneous accelerators. Based on such abstraction, developers are able to implement the appropriate subset for each XPU and to incrementally extend support as hardware capabilities evolve.

Robust OS kernels. As IoT/edge devices take on mission-critical roles (e.g., autonomous driving, industrial control, and so on), the reliability of their operating system kernels is paramount. One possible trend is building OS kernels using the Rust programming language (Rust-for-Linux), which can accommodate formal verification techniques. In practice, Rust can provide the type system that enforces crash consistency rules (e.g., "persistent objects cannot be modified outside transactions"). It reconstructs volatile state (stack) by replaying execution and bypassing already-committed transactions. Leveraging these advanced features can help enhance the OS robustness by moving "less sensitive" resource management out of the core privileged domain and using the type system to enforce memory safety at compile time. Indeed, this represents a significant challenge to the monolithic Linux kernel architecture. Therefore, OS kernels need to achieve highly efficient synchronization for distributed memory systems by removing the need for expensive synchronization (locking/invalidations) for every access.

Dynamic kernel extensibility. Complementing the safety of Rust, recent efforts such as extended Berkeley Packet Filter (eBPF) are reshaping the OS into a modular, programmable substrate. eBPF allows operators to dynamically load sandboxed, privileged logic into the kernel, e.g., custom packet filters, energy-aware schedulers, or security monitors, without the risk of crashing the system or the need for disruptive reboots. We argue that such techniques are particularly valuable for IoT/edge devices, as they effectively refactor the monolithic kernel into a microkernel-like architecture where high-performance functions are injected at runtime. For resource-constrained edge nodes, recent work [71] explores "decoupled verification" where complex safety checks are offloaded to a powerful server, allowing the edge device to load the verified bytecode with minimal overhead. This enables a fleet of IoT devices to instantly adapt their kernel-level security policies or network protocols in response to zero-day threats or changing environmental conditions. Furthermore, eBPF is also used to offload complex scheduling policies or distributed transaction logic directly into the kernel datapath, bypassing the heavy overhead of user-kernel context switches [72].

Minimal kernel design. Compared to traditional microkernel architectures, a minimal kernel retains only performance-critical and timing-sensitive mechanisms in kernel space, while aggressively minimizing kernel complexity and trusted computing base size. For example, the proposed XiUOS [73] positioned the kernel as a fast, predictable execution substrate that directly supports real-time scheduling, low-latency interrupt handling, and efficient memory management, while allowing higher-level services and policies to be modularized and extended outside the kernel when isolation or flexibility is required. This design represents a deliberate tradeoff: it sacrifices the strict service separation of microkernels in favor of tighter control over performance and determinism, which are essential for ubiquitous and resource-constrained systems.

Lightweight runtime for IoT/edge workloads. Given that UOS needs to support lightweight computing on IoT/edge devices, WebAssembly (Wasm) is emerging as a promising candidate for a universal binary format. Future operating systems like UOS are expected to manage Wasm runtimes directly, rather than traditional Linux processes, enabling millisecond-level startup times and exceptionally high deployment density. For example, VectorVisor [74] introduces a runtime that executes generic WebAssembly computations directly on the GPU. This paradigm shift leverages the inherent parallelism of the GPU for general-purpose computing beyond mere graphics rendering, achieving high isolation and superior performance for Wasm tasks at the edge. WALI [75] addresses Wasm's inability to efficiently run complex system-level software by proposing a "thin kernel interface". It constructs an efficient adaptation layer between user space and the kernel, allowing Wasm modules to execute standard Linux applications with minimal overhead.

Hardware-software co-design for isolation. As SCPS workloads essentially require more heterogeneous hardware resources like x86, ARM, and RISC-V, some new hardware isolation primitives (Intel MPK, ARM PAC, MTE, CHERI) are worth mentioning to build UOS. Recent efforts are critically evaluating their utility for isolating untrusted subsystems (e.g., third-party libraries or drivers). Future hardware must support "core-coherent" synchronization (updates to permissions are instantly seen by all cores) to enable zero-copy data passing.

3.3.2 *Efficient infrastructural supports for AI model serving*

AI models, especially LLMs, become key components of SCPS applications. As a result, the infrastructural support for serving AI models needs to be considered as a first-class citizen in future OS design.

On-cloud model serving. Currently, serving LLMs is mainly deployed on centralized cloud/servers. Kernel-level optimizations are widely employed to speed up computation [76, 77] with specialized CUDA kernels, softmax/GEMM improvements, and AMD GPU support (cf. DeepSpeed-Inference [78]). At scale, systems combine data, tensor, pipeline, and expert parallelism to efficiently utilize multi-GPU/TPU clusters [78, 79]. Beyond kernels, request batching and scheduling improve utilization [80–84]. A noticeable trend of inference acceleration is the emerging disaggregated architecture that assigns prefill and decoding to different GPU nodes, which has practically motivated a range of system-level optimizations [85, 86].

On-device model serving. While model serving targets datacenters traditionally, new work adapts to heterogeneous and resource-constrained environments on the IoT/edge devices [87–89]. Some recent research efforts focus on how to deliver LLMs to edge hardware and OS more efficiently. Specifically, there are two notable research trends. One trend is the algorithm-hardware co-design for heterogeneous accelerators. Such a trend aims to move toward deep algorithm-hardware co-design to address the extreme fragmentation of mobile system-on-chips (SoCs). Rather than treating the device as a monolithic compute resource, systems are now designed to exploit the distinct strengths of heterogeneous components, including CPUs, GPUs, NPUs, and others. Kernel-level optimization continues at the device level, including integer-only quantization for transformers [90] and NPU offloading for LLM inference [91]. Based on such optimizations, new frameworks can decompose inference tasks, offloading stable, compute-intensive phases (like prefilling) to the NPU while delegating dynamic or precision-critical outliers to the CPU or GPU. This fine-grained mapping significantly improves latency and energy efficiency, enabling complex models to run locally on resource-constrained hardware. Indeed, some resource constraints include limited memory and energy budgets for serving AI on the IoT/edge devices. The edge-cloud collaboration is a promising approach to balancing latency and accuracy [92]. Some system-level optimizations like on-device mixture-of-experts methods [93, 94] can reduce computation dynamically by activating only a small fraction of the parameters for each specific input, or address memory footprint by speculative decoding [95, 96], and selective neuron/parameter loading. I/O bottlenecks are mitigated by adaptive bit-width loading (STI [97]) and flash-aware data management [98].

3.3.3 *Intelligent and autonomous system management*

UOS should be able to continuously sense environmental changes and quickly perform adequate resource management for online adaptation. To make the resource adaptation to changes timely and efficient, it requires intelligent

techniques to automate the generation of system configurations, policies, and scheduling mechanisms, and reduce human efforts.

Data-driven system diagnosis. UOS must ensure continuous, reliable service, potentially over decades of continuous operation [69]. Achieving this requires fast and accurate system diagnosis to detect and localize bugs, errors, and failures so that corrective actions can be taken immediately. Given the complexity of modern OSs and applications, effective diagnosis demands synthesizing diverse data sources, e.g., runtime logs, code artifacts, configuration settings, and more, beyond the capacity of manual analysis. The data-driven system diagnosis offers a promising path, enabling automated bug localization and prediction [99], misconfiguration detection [100], and anomaly detection [101]. Diagnosis results can then directly inform system decisions and recovery strategies. A key challenge, however, lies in accuracy: false positives or misclassifications may themselves cause instability or failures.

"Learned" operating system. To keep pace with the speed of modern hardware and application evolution, a recent proposal, which is interesting but a bit radical, suggests the design of a "learned operating system" [102]. This proposal aims to leverage AI and machine learning techniques to automatically "learn" how to build and tune an OS. Machine learning can augment—or, in some cases, replace—several classes of traditional OS components. First, ML models can be used to dynamically tune system configurations in response to workload and environmental changes. Second, ML techniques can infer and adapt OS policies based on observed application behavior and underlying hardware characteristics. Third, and most ambitiously, ML can be integrated into the implementation of core OS mechanisms themselves, fundamentally rethinking how certain system functions are realized. There are two possible scenarios.

• **Learning policies.** Traditional resource-allocation policies often rely on heuristics or static rules, e.g., file systems may allocate contiguous storage for files in the same category, or online video platforms may provision additional storage and bandwidth for popular content. ML can enhance core OS mechanisms such as memory and file management. For example, address mapping tasks, e.g., from virtual to physical memory (via page tables) or from file name and offset to disk block addresses (via multi-level indices), are central to system performance but incur significant space and latency costs. ML-based models can reduce overhead by learning workload-specific mappings and generating customized policies. Unlike fixed-size memory pages in conventional systems, ML-driven mapping can infer variable sizes and offsets, potentially achieving a smaller footprint and faster performance than multi-level structures [102]. For instance, Lynx employs ML-based prefetching from SSDs using Markov Chains to identify I/O access patterns and estimate transition probabilities between file pages, thereby improving throughput and reducing latency [103].

• **Learning configurations.** OS configuration is notoriously time-consuming, error-prone, and requires significant domain expertise. Configurations involve numerous interdependent parameters (e.g., performance, cost, security), and tuning typically relies on heuristics, trial-and-error, or offline experiments. Once deployed, configurations often remain static, failing to adapt to evolving workloads or environments. ML offers a promising alternative by training models on diverse data sources, including code evolution, execution traces, and user behaviors, to automatically generate and dynamically update configurations in response to system changes. ML has been shown to be effective for both time-related parameters (e.g., interrupt frequency, buffer flushing) and space-related ones. Despite these benefits, deriving accurate configurations remains challenging. ML models must search across two vast spaces: the policy space (all possible configuration policies) and the resource state space (e.g., idle, busy, expired). Exploration is difficult because valid policies are rare within the policy space, while violated policies are sparse within the state space [104]. Addressing these challenges is essential for realizing ML-enabled, self-configuring operating systems.

Advances in AI techniques and the availability of "big data" from OS and compute resources have made it interesting and promising to apply AI in building and managing OS that previously relied on human efforts. However, given the complexity of OS and ubiquitous computing environments of SCPS, these approaches are still at dawn. We expect more design, development, and deployment challenges to appear.

3.3.4 *Human-centric OS primitives*

As SCPS permeates human daily life, it is expected that the UOS may evolve from only a manager of hardware resources (CPU, RAM) to an "agent-styled" steward of human resources (memory, trust, privacy, etc). A Human-Centric UOS elevates user well-being to a first-class system objective, enforcing policies that protect the user's private or sensitive data.

System-wide agentic universal intermediaries. A key manifestation of the "agent-styled" design for UOS is the integration of vision-language model (VLM) based GUI agents [89] that interact with applications through the visual interface rather than rigid APIs. This approach offers unparalleled flexibility, allowing the OS to automate

tasks across any application by "seeing" and "clicking" like a human user, bridging the gap between legacy software and modern AI workflows. However, such visual omniscience also introduces severe privacy and security risks [105]. Granting an agent the ability to read the screen (screen scraping) exposes all rendered sensitive data, including passwords, private messages, and financial details, to the model provider. Consequently, a human-centric UOS must implement visual least privilege, where the OS dynamically masks sensitive UI elements (e.g., blurring banking balances) before the screenshot is passed to the agent, or runs the VLM inference within a secure local enclave to prevent data leakage.

Contextual memory fabric and privacy. In an agent-styled OS design, user data transforms from static storage into dynamic contextual memory that fuels agent intelligence. Rather than leaving apps/agents to manage their own isolated data, the future UOS can consider aggregating fragmented digital footprints, such as app usage, location history, and communication logs, into a unified, semantic vector store as a system primitive. This memory fabric allows agents to query user context (e.g., "retrieve the document I worked on last night") via high-level APIs without requiring persistent access to raw files. Crucially, the UOS leverages this centralization to enforce retrieval-aware privacy. Acting as a secure gatekeeper, the OS sanitizes the context window before it reaches the agent, employing techniques like local-only embedding generation and semantic filtering. This ensures that agents receive only the minimum information necessary for their immediate task. As an example, MemGPT [106] proposes a hierarchical memory architecture for LLM agents that mirrors traditional OS virtual memory. It demonstrates how an OS can manage an infinite context window by paging data between fast context (RAM) and slow external storage (Disk).

Human attention management. In an agent-styled OS design, the scheduler evolves into an intelligent gatekeeper for human attention—a critical resource of humans with limited budget. Rather than passively delivering notifications, the UOS acts as a "digital butler" that continuously monitors the user's cognitive load via biometric and behavioral sensors. It dynamically negotiates with applications, enforcing policies that defer non-critical alerts during high-stress or deep-focus periods. This turns the OS into an active mediator that prioritizes human mental capacity over application demands, ensuring that digital interactions align with the user's immediate intent and physiological state. For example, InteractOut [107] demonstrates that an active OS agent can accurately discourage habitual usage (e.g., doom-scrolling) by subtly manipulating input friction, effectively acting as a steward for the user's long-term well-being goals.

4 Challenges on the quality assurance of SCPSs

Normally, SCPSs are deployed in environments that are, at best, partially known at design time and dynamically evolve. Moreover, human and/or human-related social organizations become partners (components) of SCPSs, with different stances towards technology, device, and software use, different competencies, and their own sub-objectives. Therefore, as key tasks and activities, such as perception and decision-making, are increasingly shifted from humans to CPSs, the abstraction levels of interactions between humans and CPSs increase drastically, from low-level control to joint high-level analysis, reasoning, and planning. This shift introduces a new level of system design that requires mutual introspection into state, capability, intent, and strategy development between cooperating partners, whether humans or CPSs. Additionally, the main objectives of SCPSs may be non-technical, such as social fairness, ethics, and justice, rather than the technical objectives of traditional ICT systems. All partners in the system must cooperate with each other to achieve the prescribed collaborative (social) objective(s). So, it calls for the negotiation of the system objectives and individual objectives, the seamless communication, interaction, and collaboration of partners.

4.1 Challenges and opportunities of SCPSs quality assurance

As discussed in Section 2, the distinguished features give rise to substantial challenges. In what follows, we will discuss the challenges, respectively, from modeling, specification, and assurance techniques, i.e., the major issues related to the quality assurance of SCPSs.

Modeling. In order to deal with the partial information, "high-level" interaction between human and CPSs, as well as autonomy of SCPSs, as pointed out in [108], a reference architecture model of SCPSs should possess the following capabilities.

- Capturing salient classes of system-level interactions between CPS and professional or semi-professional humans relevant for societal acceptance of such systems.
- Capturing all relevant aspects of physical and societal environments of such systems within a specified operational design domain.

• Capturing, expressing, assessing, and adapting beliefs about physical and societal environments and their confidence levels.

• Reasoning about uncertainties caused by lack of knowledge and observation noise.

• Capturing strategies for self-adaptation and self-evolution in dynamically changing contexts.

• Supporting ex-ante, on-line, and ex-post analysis of all incidents/states potentially impacting safety, trust, compliance with ethical and/or societal principles or regulations.

• Supporting seamless cooperation of mixed teams of humans and CPSs to reach shared goals within given time-frames on multiple levels of interaction.

• Capturing temporary or persistent formation of coalitions.

• Supporting analysis of realizability of system goals and synthesis of strategies to achieve such goals, possibly involving coalition partners.

• Supporting the assessment of trustworthiness of SCPSs.

• Supporting justifications for actions chosen by SCPS subsystems.

• Supporting divide-and-conquer mechanisms like separation of concerns, composition/decomposition, abstraction/refinement, to attack design complexity and improve efficiency and reliability.

• Serving as a conceptual framework for building such systems in multiple application domains through specialization.

Clearly, none of the existing models for CPS, including graphical models [109–112] and formal models [113–115], nor neural networks for AI [9, 116], nor their combination, the so-called neural-symbolic computation models like multi-agent games [117], can satisfy the above criteria. It calls for a new computation model that can combine symbolic computation, neural computation, and agentic computation, that is, novel neural-symbolic-agentic computation models. Some of the first prompts have been done, e.g., [108].

As with classic programming theories, to guarantee the trustworthiness of SCPSs, a semantic foundation is necessary. However, existing semantic models are not expressive enough for SCPSs. Unifying theories of programming (UTP) [118] can unify different programming paradigms, models, and languages in classic programming theories, and its extension to CPS called higher-order UTP (HUTP) [119] provides a uniform view of different components of a CPS represented by different models at different abstract levels. However, HUTP cannot address AI and social issues. A recent development of a semantic foundation due to Damm et al. [120] is still restrictive. As a mathematical model of SCPSs, it should possess the following properties.

• Characterizing different behaviors of SCPSs at different abstraction levels, across physical, cyber, and socio worlds and their combination, particularly, high uncertainty from physical and societal worlds.

• Modeling the interaction and communication among partners subject to partial and incomplete information; specifically, the interaction and communication topological structure could dynamically evolve and change.

• Modeling the black-box and unexplainability of AI components.

• Dealing with heterogeneity and interoperability. A complex SCPS consists of many highly heterogeneous subsystems, which may be designed using different design theories and tools, particularly those related to human behavior.

Specification. Also, none of the existing specification logics can be used to specify and reason about SCPSs, as a specification logic for SCPSs should possess the following capabilities.

• Characterizing respective technical properties related to the cyber and physical worlds like mobility, hybridization of continuous and discrete evolution, real-time, stochasticity, probability, time-delay, security and privacy, and so on, and non-technical properties related to the societal world, like fairness, rationality, justice of laws and policies, and so on, specifically, their deep entanglements.

• Uniformly specifying and reasoning qualitative and quantitative properties, and the transformation between them according to the confidence levels of humans.

• Specifying and reasoning about properties under partial and incomplete information, particularly the evolution and dynamics of ego systems and their environments.

Perception. Perception plays a key role in the design of SCPSs, which is also crucial to guarantee the trustworthiness of SCPSs, as most decisions are made based on perceived data. Traditionally, perception is done by humans or by independent instruments with human intervention. But in SCPSs, it should be shifted to the CPS level in order to obtain a quicker response by improving automation. For instance, in modern weapon systems like fighters, since the fourth generation of fighters, it has been a strict demand to have the capacity to observe, orient, decide, and act (OODA). However, perception is challenging and error-prone due to the following factors.

• Complicated behaviors of physical processes and environments, which are either modeled by complicated mathematical models such as delay differential equations, stochastic differential equations, and partial differential equations, or cannot be modeled by any physical principle at all.

• Measuring human beings' intentions, beliefs, and values. Usually, a state of an SCPS includes the sub-states of its societal components, so measuring the state of an SCPS heavily depends on the beliefs, intentions, and values of human beings, who are the partners of the SCPS. However, it is unclear how to measure them in psychology.
• Numeric error happened in sampling and representation.
• The black-box and unexplainability of AI components.
• Merging data with different formats from different sources, particularly subject to real-time constraints,
• Security and privacy.

In the literature, many research areas are paying increasing attention to this issue, including the CHI, robotics, and CPS communities. For example, in the robotics community, ROS [121] was proposed; one of its major goals is to sense, merge, and process multimodal data collected by different sensors. However, it remains a significant challenge to take human issues into account.

Simulation and testing. Because of the inherent complexity, the design of SCPSs should be model-driven, combined with data-driven, and complemented with design-by-contract. Simulation and testing will play a central role, at least for the following reasons: (i) simulation and testing can be used to validate system models, which provide a digital twin of the real-world system; (ii) simulation and testing provide a cheaper, intuitive, and efficient way to guarantee the trustworthiness of the design; (iii) simulation will be applied to predict the behavior and evolution of the system and steer the system towards its social objectives.

But the simulation and testing of SCPSs will become more difficult. These challenges encompass the need for more realistic and data-driven models, grappling with scalability concerns inherent in large-scale simulations, and the crucial task of integrating uncertainty into simulations. Some first attempts to address this issue have been made by combining with AI techniques, e.g., [64, 67] and DT-based testing [122].

Verification. Formal verification is a very important technique to guarantee the trustworthiness of systems, particularly mandatorily recommended by different standards for the design of safety-critical systems in different domains. As said before, because of the new features of SCPSs, it introduces substantial challenges in the verification of SCPSs, encompassing the following aspects.

• Incremental and/or continuous guarantee because of self-evolving and adaptability.
• Verification of black-box systems, particularly related to the non-technical properties of SCPSs.
• Efficiency and scalability. These bottlenecks of existing techniques will become more notorious because of the inherent complexity of SCPSs.

So, it calls for new verification theories and techniques for SCPSs.

Controller synthesis and decision-making. Controller synthesis and decision-making provide a correct-by-construction approach to guaranteeing the trustworthiness of SCPSs, which is also one of the major tasks of SCPS design. Most SCPSs are autonomous or semi-autonomous, being composed of autonomous or semi-autonomous partners (subsystems). This requires that any SCPS and each of its partners should have the OODA ability. That is, given an objective for an SCPS, each component of the SCPS can obtain its own sub-objective and synthesize a strategy so that these partners can cooperate with each other to achieve the objective. E.g., in self-driving, each car can be seen as an autonomous system, while other cars and the environment (including the road, pedestrians, and so on) are the adversarial components. The objective of the whole system could include safety (i.e., crash avoidance), reachability (i.e., the arrival at the target of each car), maximum throughput, etc. While the objective of an individual car may also include less driving time, a more comfortable level, etc., in addition to the overall objective. Synthesizing real-time driving strategies while balancing the global objective and the individual objectives of the partners is a challenge in the design of self-driving systems. Certainly, its correctness is also the key to the trustworthiness of self-driving, yet ensuring it is very difficult due to the following reasons.

• It depends on many open mathematical problems. Controller synthesis is one of the most important topics in classical control theory, attracting substantial attention. However, many problems remain open because they essentially correspond to open mathematical problems.
• Most of the AI components are black-box and unexplainable. This significantly complicates controller synthesis and decision-making.
• Imperfect information makes controller synthesis and decision-making more difficult.
• High uncertainty from the physical and societal world makes the problem more difficult, particularly when a system objective may be non-technical, more related to societal factors.
• Mathematically speaking, many of the SCPSs can be modeled as multi-agent systems at a very abstract level, possibly with some unknown components, such as human partners. The question of how to synthesize winning strategies remains open in the theory of multi-agent games [117].
• Data-driven controller synthesis has been widely used in classical controller synthesis but suffers from data scarcity and the limited precision of sampled data. This becomes worse in SCPSs as we pointed out in the previous

subsection.

Online monitoring and enforcement. It is widely recognized that online monitoring and enforcement serve as a crucial complement to simulation, testing, and verification in ensuring the trustworthiness of complex systems. During the development stage, testing and simulation are inherently incomplete; static analysis can only verify simple properties; and heavy formal methods, due to cost and scalability limitations, are generally applicable only to small or medium-sized systems at most. As a result, a complex system still contains some mistakes, inevitably, after deployment. Therefore, monitoring individual system executions to detect or predict abnormal behavior caused by design flaws or external attacks, and then shielding the potential exception or enforcing corrective behavior in its environment to avoid the attack, becomes practically significant. Given the high complexity of SCPSs, it is necessary to invent online monitoring and enforcement techniques for them. To this end, we need to address the following challenges.

• Specifying complex properties of SCPSs. First of all, it requires a specification logic, which can express all kinds of complicated properties uniformly, including technical and non-technical, and qualitative and quantitative. Meanwhile, it also supports efficient monitoring and enforcement algorithms; Second, it calls for techniques to extract underlying complex properties. Normally, they are provided by domain experts. Recent advances in LLMs may provide another way to extract such complex properties, particularly by automatically synthesizing specifications given by the specification logic in the previous step [123, 124].

• Efficient monitoring algorithms must determine or predict whether the considered execution has violated or will violate the specified specification. This task is particularly challenging in SCPSs due to perceptual noise and systemic uncertainties. Ensuring the real-time performance of monitoring algorithms, as well as the accuracy and interpretability of machine learning-based prediction algorithms, requires the development and application of appropriate mathematical tools.

• Efficient enforcement and reasoning algorithms are needed to correct the attacked behavior or the foreseeable unexpected behavior. The inherent complexity of SCPSs, combined with the subtleties of non-technical properties, makes enforcement highly non-trivial. Although machine learning techniques may offer potential solutions, it remains unclear how they can be reliably applied in this context.

4.2 Trends of the quality assurance of SCPSs

The general trends on SCPS quality assurance have been discussed in Subsection 2.5.4. In this subsection, we will stress some special trends, related to modeling, particularly self-evolving computing, early fault detection, verification, and monitoring, etc.

4.2.1 *Modeling self-evolving computing*

To realize the vision of SCPSs, engineering solutions that enable them to operate 24/7 and evolve autonomously in ever-changing, unanticipated conditions are required. We refer to this process as *self-evolving computing*. Self-evolving computing brings about a new conceptual thinking of software evolution [125]. When a self-evolving computing system detects a relevant, unanticipated change, it launches an evolutionary process to adapt its architecture to address it. Thereby, humans are no longer driving the evolution process, but become facilitators of evolution by offering new computing elements that self-evolving computing systems can integrate autonomously to evolve as needed.

A fundamental new concept to model self-evolution could be a "computing warehouse" that provides new computing elements that can be used by self-evolving computing systems to evolve during operation. Self-evolving computing systems can explore a warehouse catalog to find new elements and integrate them as needed. Computing warehouses can be populated with new elements by software companies, following a policy from pure private to fully open. When triggered by the detection of a relevant unanticipated change, an "evolution engine" evolves the runtime architecture model of the underlying system through an evolutionary process, thereby integrating new elements as needed. We anticipate that LLMs will facilitate self-evolution [126], particularly centered on experience accumulation, a concept that parallels self-evolution, where LLM agents continuously acquire new "skills" for emerging tasks. For instance, if LLMs identify that existing components or APIs are inadequate for unforeseen runtime conditions, they can autonomously search for and integrate new APIs (from online sources) into the adaptation space and facilitate reasoning about these components through the available API documentation. A critical gap is that self-evolution in LLM agents often relies on natural language descriptions, while self-evolving systems will reason using knowledge expressed in some form of DSML. This results in a need to define observations and skills in such a DSML. In the context of self-evolution, the DSML should support evolving itself (e.g., by introducing new actions and events) and adapting the corresponding reasoning methods.

4.2.2 *Early fault detection*

In addition, SCPSs are complex, critical, and difficult to develop. Hence, we emphasize the importance of early detection of faults, hazardous behaviors, and extreme potential scenarios, which must be considered from the very beginning of design and development. A key challenge is that automated analysis, verification, simulation, and test generation techniques and methodologies rely on well-structured and even formalized information and, indeed, on resolution and concretization of abstractly defined entities. LLMs can detect issues in natural language requirements and system descriptions, but this addresses only a small part of the task of early fault detection. Consequently, we propose (see, e.g., [127]) an architecture for models and tools that can help overcome such barriers and enable simulation, systematic analysis, and fault detection and handling early in the development of such systems, when specifications may still contain abstract, vague, and inconsistent elements, and before concrete implementation decisions are made.

The main innovations are (i) allowing the embedding of NL specifications within elements of otherwise standard models and specification formalisms, while deferring the interpretation of such NL elements to simulation, analysis, and validation time; (ii) a focus on early formalization and documentation of tacit assumptions and interdependencies.

As a (naïve) example, consider the requirements to implement the traffic rules "never enter an intersection when the light is red" and "always obey the police instructions". Automated tools will be able to quickly recognize the potential for conflicts between these rules and ask for a method to resolve them; they will also be able to use their domain knowledge to build in advance a comprehensive list of applicable police instructions and the ways by which they may be communicated.

The approach is facilitated by combining newly specialized tools with standard development and verification facilities and with the inference and abstraction capabilities of LLMs and associated AI techniques. The unique contribution of the amalgamation of formal models and NL is as follows. The formal part can provide modularization and encapsulation of many entities, which are otherwise vague when in NL, as well as many basic relationships between them, such as containment, sequential ordering, parallelization, disjointness, etc. Powerful NL processing and analysis based on extensive domain knowledge (as afforded by LLMs and their future replacements) can concretize abstract concepts, and when doing so still maintain multiple alternatives, refine the various relationships, and discover many hidden, tacit, assumed, or forgotten interdependencies between such entities. The analysis of models enriched in this manner can then be carried out by LLMs directly, or by formal, traditional tools (with proper enhancements), using the LLM-interpreted and enhanced model as a concrete formal input.

4.2.3 *Verification and monitoring*

Verification theories based on neural-symbolic-agentic computation. Verification techniques at a very early stage are mainly based on symbolic computation, particularly either based on logic proof systems or their decision procedures like theorem proving, SMT, model-checking, or based on computer algebras like quantifier elimination and rewriting systems, and so on. Later, since the end of the last century and the beginning of this century, more and more concepts related to continuous mathematics have been introduced into computer science to deal with more complex systems like embedded systems, CPS, etc. Consequently, more verification techniques based on numerical computation have been developed, particularly optimization techniques. In fact, optimization-based constraint-solving techniques have become promising in the verification of programs, probabilistic programs, and CPS. In recent years, a further development towards this trend is so-called neural-symbolic computation to build verification theories for AI-enabled systems. For verification, we may need to go further to take an agentic issue into account. This means that we need to consider theories that can express symbolic, numeric, neural, and agentic constraints, as well as their combinations, and to invent efficient algorithms to solve such constraints or logical proof systems to reason about them.

Uncertainty. Our universe is inherently stochastic, uncertain, and even chaotic. Real-world computing systems can hardly, if ever, be deployed in a fixed, closed, and perfectly predictable environment. This limitation is particularly pronounced in SCPSs [108, 120, 128], which typically exhibit the following characteristics.

- *Exogenous uncertainty*, arising from their operation in open, dynamic environments that are inherently unpredictable, difficult to characterize fully, and often marked by discrepancies between design-time requirements/assumptions and runtime realities—in fact, the environment per se can be highly volatile or even (partially) unknown.
- *Endogenous uncertainty*, induced by the ever-increasing integration of data-driven, learning-enabled, and societal components that exhibit limited predictability and interpretability.

As a consequence, specifying, analyzing, and verifying safety-critical SCPSs subject to substantial uncertainty—thereby ensuring their safety, reliability, and effectiveness—remains a significant challenge in computer science, necessitating new theoretical foundations and engineering methodologies.

Seamless integration of the verification of qualitative and quantitative properties. As argued previously, most of the properties of SCPSs can only be guaranteed in the sense of PAC because of the inherent uncertainty of SCPSs, as well as verification efficiency and scalability. However, we have to guarantee the absolute correctness of some core safety-critical components. So, it calls for new verification methods that integrate qualitative and quantitative approaches and can seamlessly switch between them based on confidence levels.

Integration of different analysis and verification techniques. Because of the inherent complexity of SCPSs, it calls for the integration of different testing, simulation, analysis, and verification techniques so that one can make a trade-off between cost and verification strength while ensuring the trustworthiness of SCPSs. At least, it should integrate the following techniques: testing and simulation; lightweight verification, such as type checking, for rapid validation; midweight verification, such as bisimulation, for behavioral equivalence; and heavyweight verification, such as deductive reasoning with AI-assisted invariant generation. These address varying levels of complexity, from static properties to dynamic behaviors.

4.2.4 *The marriage of formal methods and AI*

AI is likely to become a key technique in the design and maintenance of SCPSs, from specification extraction to model identification to system synthesis to system self-evolution and adaptability, and so forth. Moreover, as we discussed previously, formal methods are indispensable for the quality assurance of SCPSs. So, combining formal methods and AI should be investigated with the highest priority in building the foundations for SCPSs, as they are complementary to each other.

On the one hand, AI can enhance the capabilities of a system under design, but it also raises the trustworthiness problem. To increase the trustworthiness of AI components using formal methods, some recent attempts have focused on code generation with correctness guarantees using refinement calculus [129], neural-symbolic agents [130], and the transformation from informal to formal models [123, 131]. These studies have been further extended to runtime assurance [64, 67, 132], safety guardrails for LLMs [133], and privacy preservation AI [134].

On the other hand, the bottleneck of existing verification techniques lies in efficiency and scalability. In recent years, some attempts have been made by exploiting AI to improve the efficiency and scalability of formal modeling and verification, e.g., learning-based model identification [135, 136], invariant generation using data-driven for programs and systems [137, 138], improving the automation of theorem proving by LLM [139, 140], program synthesis by LLMs and data-driven [141, 142], and so on.

5 Challenges in different domains

Although SCPSs across different domains share common characteristics, they exhibit significant differences in several aspects, including: the types of human/societal entities, cyber resources, and physical entities involved; the integration and interaction modes between different types of resources; the computing and network platforms supporting socio-cyber-physical integration; and the overall trustworthiness requirements of the systems (e.g., privacy, real-time performance, reliability, cybersecurity). Consequently, the current development status and challenges of SCPSs differ markedly across these domains.

In this section, we will specifically address the challenges faced by SCPSs in three domains: intelligent connected vehicles, smart manufacturing, and smart city. These three fields are selected primarily for two reasons. On the one hand, they represent substantial sectors that are of great significance to economic and social development. On the other hand, each domain exhibits distinct characteristics, especially in terms of how socio-cyber-physical integration is implemented: intelligent connected vehicles encompass three domains—intelligent vehicle control, intelligent driving, and intelligent cockpit—each with different levels of intelligence and safety/reliability requirements, while also showing a trend towards cross-domain integration and resource sharing; smart manufacturing demands both strong real-time performance and reliability, as well as high flexibility and adaptability in production lines; smart cities involve a wide range of urban infrastructure and equipment services, entail extensive social collaboration, and can themselves be divided into multiple tiers according to the scale of the region.

5.1 Intelligent connected vehicles

With the advancement of computer software and hardware, AI, and network communication technologies, intelligent connected vehicles (ICVs) have evolved from traditional mechanical and electronic products into a new generation of intelligent terminals that are deeply integrated into a broader ecosystem of vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-cloud (V2C) communications. This connectivity enables cooperative perception, coordinated decision-making, and collective intelligence, transforming each vehicle from an isolated intelligent entity into an interconnected node within the mobility network. Complementing this connectivity, the paradigm of "software-defined vehicles" has also become a universally recognized development direction in the industry. For fuel-powered vehicles, such as Volvo XC90 and S90, the total amount of software source code can reach 100 million lines [143]. In contrast, modern electric vehicles represented by Tesla, NIO, XPeng, and Li Auto are more aligned with the concept of "software-defined vehicles", i.e., using software to manage complexity.

ICVs consist of three intelligent domains: intelligent vehicle control, intelligent driving, and intelligent cockpit. Among them, the intelligent vehicle control domain coordinates and manages the chassis, powertrain, and by-wire systems (e.g., steering/braking/suspension) through electronic control units (ECUs), enabling precise adjustment of vehicle dynamic performance and redundant safety control. The intelligent driving domain perceives environmental information via on-board sensors (e.g., cameras, radars, LiDAR), then makes driving decisions and plans based on the vehicle's own status, and controls vehicle operation through commands such as acceleration, deceleration, and steering. Moreover, through V2V and V2I communication, this domain can exchange real-time information with surrounding vehicles and infrastructure, enabling cooperative collision avoidance, traffic flow optimization, and collective perception beyond the line-of-sight of any single vehicle. The intelligent cockpit domain realizes human-vehicle interaction with drivers and passengers through voice, touchscreens, augmented reality head-up displays (AR-HUD), while providing in-vehicle audio-visual entertainment integration. Meanwhile, through V2C connectivity, it can access cloud-based services, real-time navigation data, and personalized infotainment content, enabling seamless transitions between in-vehicle and cloud-based user experiences. All these domains exhibit strong characteristics of SCPSs. Specifically, the intelligent vehicle control domain highlights the precise control of vehicle moving components by software algorithms and AI models; the intelligent driving domain emphasizes data-driven vehicle and environmental perception, decision-making, and control; the intelligent cockpit domain focuses on human-machine interaction and users' control and operation of cockpit equipment and functions.

Currently, the ICV industry is characterized by three major trends. First, vehicles are evolving from isolated systems into fully connected nodes within an intelligent transportation ecosystem. This connectivity spans multiple dimensions: V2V for collaborative perception and platooning, V2I for traffic-signal optimization and infrastructure interaction, and V2C for cloud-based data fusion, model updates, and fleet-level intelligence. Second, the vehicle's electronic and electrical architecture is moving toward centralization. The three domains (intelligent vehicle control, intelligent driving, and intelligent cockpit) are gradually integrating into a central computing platform. Meanwhile, vehicles increasingly rely on cloud platforms to support intelligent capabilities and enable over-the-air (OTA) updates. Third, various AI models (including LLMs and specialized small models) are being deployed in vehicles to enable functions such as end-to-end autonomous driving, intelligent voice interaction, and cockpit control. These three trends reflect the typical characteristics of modern SCPSs: being software-defined, networked/connected, and intelligent. Among them, the development trends of centralized architecture and vehicle-cloud integration mean that software will play an increasingly important role in defining the vehicle's overall electronic and electrical architecture and managing complex systems. At the same time, design principles such as hierarchical decoupling and cross-domain sharing have gained growing recognition. Although the software-defined paradigm, connectivity, and intelligence have become universally acknowledged development trends, the technology and industrial development of ICVs still face multiple challenges from the perspective of socio-cyber-physical systems.

• **Challenges in safety and real-time performance.** ICVs have high requirements for safety and real-time performance in environmental perception, driving decision-making, and vehicle control. SCPSs defined by software require hierarchical decoupling between software and hardware, as well as between software layers at different levels. This inevitably introduces delays and uncertainties. Delays in resource provisioning can lead to suboptimal performance or worse system failures in autonomous vehicles. Furthermore, AI models are being deployed and applied in vehicles at an increasing rate, while end-to-end large autonomous driving models have emerged as a new trend. However, the uncertainty, inexplicability, and unreliability of AI models may become new potential safety hazards.

• **Challenges in non-unified standards and specifications.** For a long time, the automotive industry has widely adopted distributed architectures and corresponding supply systems. Numerous Tiers 1 and 2 suppliers provide original equipment manufacturers (OEMs) with various components and related software-hardware solutions.

These components use different hardware, operating systems, middleware, and communication protocols. Amid the development trends of centralized architecture and software definition, the ICV industry needs to gradually establish a unified system of standards and specifications, which poses significant challenges in terms of technology and ecology.

• **Challenges in hierarchical decoupling and system complexity.** The AI model architectures that enable autonomous driving are evolving from modular to end-to-end. However, in vehicle-cloud collaboration scenarios, the software system must not only have highly modular atomic components to support flexible configuration but also ensure real-time performance and determinism during cross-layer and cross-domain function invocations. This architectural model of "coexisting decoupling and coupling" gives rise to issues such as inconsistent interfaces, uncertain timing, and functional redundancy. A key challenge in realizing vehicle-cloud integration is designing automotive software that supports rapid iteration and flexible combination while meeting automotive-grade real-time performance and functional safety requirements.

• **Challenges in AI software-hardware co-design.** ICVs need to rely on automotive-grade dedicated AI chips (e.g., GPUs, NPUs, FPGAs) to achieve efficient and low-latency AI model inference acceleration. Furthermore, they require rational CPU collaboration and heterogeneous computing resource scheduling to ensure model deployment performance while realizing efficient sharing and flexible allocation of computing resources. This presents challenges to the software-hardware co-design of AI functionalities. In addition, one particularly promising direction is embedding AI capability as an operating system service rather than treating it as an application-level feature. By moving AI functionalities into the core system stack, devices can achieve greater efficiency, reduced latency, and seamless integration with other system components.

• **Challenges in trustworthiness.** The connectivity inherent to ICVs, through V2V, V2I, and V2C communications, not only enables cooperative functionalities but also significantly expands the attack surface and introduces new failure modes. Distributed propagation of erroneous information across the network may lead to large-scale hazards, such as cascade collisions. Therefore, trustworthiness is the foremost concern in deploying ICVs. These vehicles operate in dynamic, unpredictable environments where real-time decision-making is crucial. The introduction of autonomy and interconnectivity increases the potential for cascading failures, where a single erroneous decision or communication failure can propagate across a network, leading to large-scale hazards. In addition, ICVs are highly vulnerable to cyberattacks, including data spoofing, denial-of-service (DoS) attacks, and adversarial manipulation of AI components. Given their reliance on wireless communication, robust cryptographic protocols and intrusion detection systems are imperative. Therefore, achieving the trustworthiness of ICVs requires a rigorous, multidisciplinary approach incorporating formal verification, security analysis, and real-world testing under a unified semantic framework.

5.2 Smart manufacturing

Industrial manufacturing capability is one of a country's core competencies, reflecting its strength in numerous disciplines such as mathematics, physics, chemistry, materials science, mechanics, electronic engineering, and computer science. With the increasing diversification and complexity of production objects, computer technology, especially computer software and AI, plays an increasingly critical role in industrial manufacturing.

The core elements of industrial manufacturing are typically summarized under the "4M1E" framework: Man (human resources), Machine (equipment), Material (raw materials), Method (processes), and Environment (production conditions). The modern industrial manufacturing environment is a typical SCPS, where software has deeply penetrated all links of manufacturing and plays a pivotal role [144]. First, production equipment is gradually defined by software. The flexibility of software endows equipment with multiple functions and enhances its expandability. Second, for complex production objects composed of numerous parts and materials (e.g., smartphones), software can be used to plan production schedules, manage process flows, and coordinate rational process scheduling. This improves the degree of production automation, reduces material consumption, and enhances the efficiency of warehousing and logistics. Third, industrial manufacturing lines require continuous operation, making equipment handling and preventing failures and quality issues more important. By monitoring and analyzing data on equipment status and processing quality, software systems can provide timely decision support for the operation, maintenance, and management of manufacturing lines [4]. In addition, humans still play an irreplaceable role in production and management, including manual deployment in key processes and necessary interactions between humans, equipment, and software systems [145].

The intelligence of manufacturing systems can not only significantly improve manufacturing efficiency but also enhance system flexibility to better meet diversified production needs. After years of exploration, the Industry 4.0 framework proposed by Europe and the United States has gradually matured and is moving towards a higher level

of intelligence. In the Industry 4.0 era, software is the core driver of manufacturing process automation. In the future, the role of AI will become even more prominent, while software will remain the foundation and carrier for AI operation [146].

The development of smart manufacturing is a gradual evolutionary process. On the one hand, advanced manufacturing systems themselves are extremely complex, often requiring the integration of equipment and technologies from multiple suppliers, which cannot be achieved overnight [4]. On the other hand, industrial equipment involves high investment costs. Therefore, using software to decouple process flows from equipment capabilities and fully tap into the potential of existing equipment has become an effective path to accelerate the digital and intelligent transformation of the manufacturing industry.

Software-defined manufacturing systems can effectively address characteristics such as system complexity, diversity of production objects, precision of process flows, and continuity of production. However, the design and implementation of smart manufacturing systems still face multiple challenges from the perspective of SCPSs.

• **Challenges in interoperability.** In a SCPS, manufacturing elements (humans, equipment, materials, processes, and environment) must form a closely collaborative whole to realize the value of smart manufacturing. However, production equipment on the line may be highly customized, with significant differences in interfaces, control methods, and capabilities. Middleware and production management software are also deeply customized according to the manufacturing environment, resulting in large variations in functions and interfaces. This high degree of customization at the hardware and software levels hinders the integration of equipment from different suppliers and of different models, and impedes the continuous evolution of manufacturing systems. Establishing open, unified, and sustainably evolvable interoperability standards is not only a prerequisite for ensuring the interconnection between equipment and software but also a key to promoting the healthy development of the industrial ecosystem.

• **Challenges in real-time performance and reliability.** In advanced manufacturing systems, production decisions depend on multiple factors, such as real-time production demands, equipment status, and characteristics of production objects. Efficient, reliable, and low-latency data collection and processing are essential for the agile response of manufacturing systems. In scenarios such as high-precision machine tool control and flexible production line switching, the real-time performance and reliability of the system directly determine product quality and production efficiency. To improve the versatility and flexibility of equipment, SCPSs often use software abstraction to achieve the diversity of equipment capabilities and the decoupling of process flows. However, this layered abstraction increases system complexity to a certain extent and affects system real-time performance and reliability. Therefore, finding a balance between abstract decoupling and real-time reliability has become an urgent issue to be solved in smart manufacturing.

• **Challenges in system complexity.** Complexity is an inherent attribute of smart manufacturing. A complex manufacturing process involves multi-dimensional interactions between humans, machines, materials, and the environment, and the quality of each link in the production flow determines the quality of the final product. Consequently, the production system is inevitably a complex system integrating a large number of equipment and software. It poses enormous challenges in designing, implementing, and maintaining such a complex system with hardware-software collaboration, and ensuring its reliable, stable, and efficient operation. From the perspective of SCPSs, this is an interdisciplinary and cross-level systems engineering problem, which requires collaborative innovation in architecture design, hardware design, software design, interface design, and human-machine interaction.

• **Challenges in diversity.** Smart manufacturing systems need to be diverse to cope with the rapid changes in market demands and the differentiated design of products. To support low-volume and high-variety production, the manufacturing system must have high flexibility to achieve rapid switching of production modes. To resolve the contradiction between the diversity of production demands and the immutability of physical equipment, how to realize the diversification of equipment production capabilities by achieving modular design, platform-based architecture, and standardized interfaces is another major challenge that SCPSs must address.

• **Challenges in self-evolving.** Manufacturing systems are required to operate 24/7 and evolve autonomously under ever-changing and unanticipated conditions. In this case, humans are no longer driving the evolution process but become facilitators of evolution by offering new computing elements that self-evolving systems can integrate autonomously to evolve as needed. A resource warehouse can be constructed to provide new computing and operating elements that can be used by self-evolving systems. We anticipate that LLMs will facilitate self-evolution [126], particularly centered on experience accumulation, a concept that parallels self-evolution, where LLM agents continuously acquire new "skills" for emerging tasks.

5.3 Smart city

Smart city represents a pivotal measure to build new competitive advantages for cities in the future. From urban e-government and community management services to smart home systems for households, smart city applications across different regional scopes and types are leveraging digital technologies to gradually enhance social governance capabilities and improve people's quality of life.

Advancements in mobile communication, the IoT, and other technologies have enabled smart city systems to transcend the boundaries of traditional management platforms centered on information processing. These systems have gradually evolved into ubiquitous software systems that are deeply coupled with the physical world. Such systems can not only connect and control various urban physical devices in real time but also seamlessly integrate into the response workflows of professional service personnel. Resources or entities from the three-dimensional spaces of socio (human services), cyber (AI computing services), and physical (devices such as sensors and actuators) collaborate and coordinate on demand. Through software-defined approaches, a ubiquitous application model is formed that integrates online and offline functionalities. In this case, smart cities can be regarded as a typical scenario for in-depth socio-cyber-physical integration.

Different smart city scenarios achieve their respective goals and requirements through software intermediaries such as cloud-based management platforms and personal mobile applications. Urban governance systems collect real-time data on transportation, environment, and other aspects via IoT devices, and provide managers with precise decision support through algorithmic models [8]. Commercial systems like food delivery services respond to order requests from user terminals, match merchants with delivery riders via platform algorithms, and track delivery routes in real time using positioning devices [147]. Park energy-saving systems monitor energy consumption data through sensors, optimize equipment operation via intelligent control algorithms, and allow managers to remotely intervene [148]. Community security systems perceive the environment using devices such as surveillance cameras and smart access controls, issue alerts for anomalies through AI recognition technology, and link property personnel for rapid responses [149]. Smart home systems receive user commands via voice assistants and smart appliances, and achieve device interconnection through a home control hub [5]. All the above systems fully demonstrate the characteristics of SCPSs. They achieve scenario-specific goals and strive to optimize resource utilization by orchestrating resource capabilities and scheduling resource instances.

Currently, smart cities are gradually evolving from digitalization to intellectualization. First, AI technology is driving the enhancement of smart city system capabilities. For instance, technologies such as multi-modal data fusion, real-time algorithm optimization, and federated learning enable in-depth analysis of urban operation data, promoting the evolution of urban governance from passive response to proactive prediction. Second, cross-domain integration in smart cities is breaking down data and collaboration barriers. For example, the construction of unified data platforms and blockchain technologies facilitates cross-domain data interconnection, driving the upgrade of urban systems from siloed development to holistic collaborative governance.

Smart city systems are expanding into an increasing number of fields. However, the design and implementation of smart city systems still face multiple challenges from the perspective of SCPSs.

• **Challenge in unified access and management of heterogeneous resources.** Physical devices such as sensors and actuators come in diverse brands and models, with varying communication protocols and data interfaces. Additionally, online systems or services from different vendors adopt vastly different data storage and interaction methods. To effectively manage heterogeneous resources, SCPSs must address issues such as protocol conversion and interface adaptation in a unified manner when integrating these resources. Otherwise, unified resource allocation and efficient utilization cannot be achieved. On the other hand, a single, centralized architecture is unlikely to meet the requirements for managing disaggregated resources. Therefore, a hierarchical design, combined with control and data plane separation, would be necessary to handle the system scale and complexity.

• **Challenge in flexibility and reliability of system execution.** Many smart city systems solidify the collaboration processes of socio-cyber-physical resources and their instances through predefined rules or pre-coding, failing to account for the real-time status of resource instances. This leads to application failures in complex scenarios, for example due to unavailable resources. To cope with the growing complexity of demands, SCPSs demand a new form of autonomous resource provisioning and scheduling, which can dynamically plan resource usage using intelligent methods, monitor application execution, and realize adaptive adjustments to execution processes or resource binding on demand.

• **Challenge in in-depth integration of human/societal resources.** In SCPSs, social resources, especially human roles, are mostly limited to passive cooperation, e.g., completing tasks through crowdsourcing or relying on professional personnel to provide services. This results in a relatively single form of participation of human/societal entities in smart city scenarios. To offer rapid and adaptive responses to emergent societal demand, SCPSs should

seamlessly integrate with digital social infrastructures and evolve to support intuitive human-in-the-loop interactions. The systems need to provide convenient participation channels and incentive mechanisms, activate the value of human intelligence, and convert human experiential judgment and decision-making capabilities into key factors for improving the efficiency of smart city systems.

• **Challenge in multi-system organizational structure.** Currently, smart city systems with an inherent hierarchical structure (city-community-building) remain relatively independent. This prevents the smooth downward transmission of management instructions from higher levels and the seamless upward reporting of data information from lower levels. To improve the overall data interconnection of smart cities and enhance collaboration and response efficiency, SCPSs need to connect with each other through a hierarchical organizational structure. This will unblock links such as resource capability invocation and data interaction, enabling the overall improvement of smart city applications.

• **Challenge in security and resilience.** As SCPSs become increasingly decentralized, ensuring secure communication and safeguarding against cyber threats is critical. Virtualized resources, while offering flexibility, also broaden the attack surface. Autonomous scheduling must therefore integrate security considerations into resource allocation decisions. New fault tolerance mechanisms that take the heterogeneous resources into account also need to be considered.

5.4 Future prospects

Guided by the core concept of self-evolving computing and analyzed through the three dimensions towards agentic SCPS (i.e., capability, agency, and form), we present a unified outlook on the future development of three key SCPS domains: intelligent connected vehicles, smart manufacturing, and smart city. Their common evolutionary goal is to construct a new generation of infrastructure centered around the ideas of "computing warehouse" and an "evolution engine", with the purpose of supporting continuous autonomous learning and collaboration within open environments.

5.4.1 *Intelligent connected vehicles*

Intelligent connected vehicles will evolve from enclosed automation to cognitive collaborators in open environments.

Next-generation intelligent driving systems will transcend reactive driving based on fixed rules (e.g., lane keeping, emergency braking). The system may possess proactive predictive capabilities (e.g., anticipating the intentions of other vehicles) by combining neural network-based perception and symbolic knowledge based on world models. Further breakthroughs may be made by acquiring reflective capability: the system can assess the confidence and potential risks of its own AI decisions in real-time and make appropriate adaptations as required. On the other hand, the vehicle will become a node within a collaborative network of social, cyber, and physical resources. When its local capabilities prove insufficient for the current scenario, it can actively initiate collaboration with other vehicles, roadside units, or the cloud—such as joint perception, shared computing, or cooperative decision-making—to handle long-tail scenarios like extreme weather or complex accidents.

Smart vehicle software in the future will closely integrate efficient symbolic control code, data-driven neural network models, and natural language prompts guiding high-level instructions. Symbolic code ensures the determinism and safety of critical control loops, e.g., implementing anti-lock braking, executing fault-detection routines in redundant steering controllers; neural networks handle non-deterministic tasks like perception, prediction, and planning, while end-to-end models directly generate control signals from sensor inputs; natural language is used to specify high-level goals, task constraints, and query API documentation related to new skill integration from the computing warehouse. These different types of components are organized in a hybrid architecture and dynamically managed by an evolution engine. This design ensures that the system meets automotive-grade real-time and reliability requirements and at the same time possesses the capability to autonomously discover, verify, and integrate new computing elements (e.g., updated neural network models) from the cloud warehouse, achieving continuous self-evolution within safety boundaries.

Smart vehicles will progress from automatic execution of predefined policies to autonomous independent decision-making in dynamic traffic environments. Leveraging the cognitive abilities developed based on LLMs, the agents in the vehicle can not only know the "brake" command but also comprehend its underlying goals of "avoiding collision" and "ensuring safety". This enables them to creatively utilize remaining capabilities based on high-level goals when sensors/actuators partially fail, for example by switching between various perception information sources and decision-making models. Finally, vehicles will become deeply organized as part of the traffic flow, functioning as schedulable mobile agents participating in global road network optimization and emergency dispatch to reduce energy consumption and improve traffic flow efficiency.

5.4.2 *Smart manufacturing*

Smart manufacturing systems will evolve from rigid production lines to self-evolving socio-physical ecosystems.

The implementation form of smart manufacturing systems will tightly integrate symbolic process rules and scheduling logic, neural network-driven visual inspection and parameter prediction models, and LLM-based natural language understanding and reasoning used to define complex production tasks and constraints. For example, when the evolution engine detects new materials or processes it cannot handle, it can use natural language descriptions to search for relevant knowledge components (e.g., new control parameter packages or simulation components) from the computing warehouse. These components are then tested and integrated into the system to support the new production requirements. Ultimately, successful experiences are extracted and accumulated as reusable standardized "skill" modules, enriching the computing warehouse ecosystem.

Smart manufacturing systems will move beyond reactive monitoring and control of production equipment. Through multimodal data fusion, they can proactively predict production quality and supply chain fluctuations and adjust production line configurations and scheduling plans accordingly. Their key evolution is acquiring systemic reflective capability, which further empowers the system with mature self-adaptation and self-evolving at runtime. For example, by analyzing production deviations, the system can not only adjust parameters but also diagnose root causes (e.g., insufficient process models, inaccurate tool wear models, or unaccounted-for new material properties) to restore normal operation through self-healing or enhance operational performance through self-optimization. Subsequently, it can leverage resources from the computing warehouse on demand to initiate an evolution process, such as integrating a new material processing simulation model or optimization algorithm.

Smart manufacturing systems will upgrade from automatic execution of fixed processes to autonomous operation with dynamic scheduling in response to order disruptions and equipment failures. Through the understanding of production plans and process documentation based on LLMs, the system will possess autonomous operation abilities. For example, the system can understand the trade-offs among multiple abstract goals like "delivery urgency", "energy consumption", and "worker workload" and formulate appropriate concrete execution plans. Ultimately, the entire workshop will evolve into a so-called "lights-out factory" (or "dark factory"), i.e., fully automated production facility where manufacturing processes can run completely independently in the dark. The whole system will exhibit highly organized characteristics: humans, machine tools, robots, AGVs, and management systems all act as agents. They autonomously form teams, negotiate tasks, and dynamically reconfigure physical facilities, and information flows around high-level business goals like "efficiently completing the trial production of a new product".

5.4.3 *Smart city*

Smart city systems will evolve from information integration to organisms of agile human-machine-thing fusion.

The technological form of smart city systems is inherently hybrid. Symbolic rules and strategies ensure the transparency and controllability of core governance logic; neural network models handle complex recognition and prediction like public safety risk and traffic prediction; natural language becomes the core medium connecting humans with the system and defining high-level user goals. This hybrid architecture demands robust cross-domain interoperability and self-evolution capabilities from the system. Thus, a unified domain-specific modeling language (DSML) is needed to describe resources, capabilities, tasks, and collaboration protocols. When the system needs to integrate a new type of IoT device or respond to a novel type of public event, the evolution engine can leverage LLMs to understand its natural language description, match or combine corresponding driver, model, and collaboration process components from the computing warehouse, and formally encapsulate and integrate them via the DSML, enabling seamless expansion of the city's digital twin and governance capabilities.

Smart city systems will transcend reactive responses based on various events (e.g., traffic accident alerts and citizen requests for assistance). Through cross-domain data fusion, they will achieve proactive prediction of the urban operation status and potential risks such as traffic congestion and public safety risks. Underpinning this is the critical capability for city-scale reflective analysis. It enables the ongoing assessment, via simulation, of the prospective impact that major event plans or new traffic policies may have on multidimensional urban performance indicators. In the event of unforeseen cross-domain chain events (e.g., a major public event during extreme weather), the relevant municipal, transportation, and emergency response agents can collaboratively establish a temporary "virtual organization". Guided by shared situational awareness and dynamically evolving collaboration protocols, the "virtual organization" autonomously adapts resource allocation and coordination strategies.

Smart city systems will progress from automation of single service processes to autonomous integration of cross-domain operations. By leveraging LLMs' cognitive understanding of social signals such as citizen demands and social media sentiment, the system can gain insights into latent social needs and risks. The ultimate goal is achieving

deep cross-domain organization among citizens, businesses, social organizations, and professional teams: they are no longer merely passive service recipients but become "social computing elements" that can rapidly self-organize into temporary task alliances based on emergent needs (e.g., local emergency rescue), achieving efficient and flexible mobilization and combination of resources and capabilities.

5.4.4 *Summary*

In summary, all three domains point toward a common future for SCPSs: they will evolve into societies of agents implemented in hybrid forms, endowed with higher-order capabilities, and exhibiting a high degree of agency. The human role will shift from direct controller of the system to a guide of "evolution" and a cultivator of the "computing warehouse" ecosystem. Through the "self-evolving computing" framework, systems can autonomously accumulate new knowledge components and undertake self-renewal within a unified semantic framework, ultimately enabling sustainable symbiotic development with society and the environment.

Meanwhile, it is crucial to recognize that as these SCPSs grow increasingly autonomous, the corresponding need for regulatory oversight becomes equally critical. A paradoxical relationship emerges: the greater the autonomy, the more indispensable governance becomes. While a full treatment of this autonomy-governance balance is beyond the scope of this paper, we expect that many regulatory requirements and mechanisms, originally designed for human societies, will gradually be integrated into the very fabric of these SCPSs. This integration is essential to ensure that the overall operation of the systems remains aligned with overarching human goals and values.

6 Conclusion

"Software is eating the world", but software engineering is falling behind. Three-quarters of a century after the creation of modern computers and three decades after the widespread adoption of the Internet, we are entering a new era in which innovative applications are built on the synergy of resources from cyber, physical, and social spaces. Yet, despite the ad hoc successes, these systems stretch well beyond the scope of traditional software engineering practiced in the past fifty years. Additionally, AI breakthroughs over the last decade have significantly enhanced software capabilities but have also introduced fundamental challenges to conventional software engineering.

In this article, we have examined challenges and potential directions for future engineering of SCPSs from the perspectives of development paradigm, runtime support, quality assurance, and domain applications. We believe that proactive and sustained research toward a new software paradigm for SCPSs is urgently needed, rather than passively waiting for it to emerge from practice. This is not only about advancing science and technology, but also, perhaps more importantly, about the fact that SCPSs are so deeply embedded in our world and society that the way these systems are created and used will *define* our future. We call for collective efforts from academia, industry, government, and society to investigate innovative software solutions for SCPSs that extend beyond AI alone, enabling us to address the fundamental challenges of the brave new world.

Acknowledgements This work was supported by China Computer Federation. AI tools, including LLMs, were used to check and fix language issues in the preparation of this article.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1 Broy M, Schmidt A. Challenges in engineering cyber-physical systems. Computer, 2014, 47: 70–72
2 Nuseibeh B. Engineering within boundaries when software has none. IEEE Trans Software Eng, 2025, 51: 677–680
3 Charette R N. Puncturing pernicious project pufferies. Computer, 2018, 51: 78–83
4 Törngren M, Sellgren U. Complexity challenges in development of cyber-physical systems. In: Principles of Modeling. Cham: Springer, 2018. 478–503
5 Domb M. Smart Home Systems Based on Internet of Things. London: IntechOpen, 2019. 1–13
6 Lee E A. Determinism. ACM Trans Embed Comput Syst, 2021, 20: 1–34
7 Jagtap D, Bhaskar N, Pannuto P. Century-scale smart infrastructure. In: Proceedings of the Workshop on Hot Topics in Operating Systems. New York: Association for Computing Machinery, 2021. 72–78
8 Sarrab M, Pulparambil S, Awadalla M. Development of an IoT based real-time traffic monitoring system for city governance. Glob Trans, 2020, 2: 230–245
9 LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521: 436–444
10 Xi Z H, Chen W X, Guo X, et al. The rise and potential of large language model based agents: a survey. Sci China Inf Sci, 2025, 68: 121101
11 Bubeck S, Chandrasekaran V, Eldan R, et al. Sparks of artificial general intelligence: early experiments with GPT-4. 2023. ArXiv:2303.12712

12 Shanahan M. The Frame Problem. Stanford: Metaphysics Research Lab, Stanford University, 2016
13 Geisberger E, Broy M. Living in a Networked World: Integrated Research Agenda, Cyber-Physical Systems. München: Herbert Utz Verlag, 2015
14 Wirth N. A brief history of software engineering. IEEE Ann Hist Comput, 2008, 30: 32–39
15 Mei H, Lü J. Internetware: A New Software Paradigm for Internet Computing. Singapore: Springer Singapore, 2016
16 Mei H, Huang G, Xie T. Internetware: a software paradigm for internet computing. Computer, 2012, 45: 26–31
17 Jackson M. The world and the machine. In: Proceedings of the 17th International Conference on Software Engineering. New York: Association for Computing Machinery, 1995. 283–292
18 Ma Y, Zhou G, Wang S. WiFi sensing with channel state information. ACM Comput Surv, 2020, 52: 1–36
19 Baresi L, Ghezzi C. The disappearing boundary between development-time and run-time. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. New York: Association for Computing Machinery, 2010. 17–22
20 Lu J, Xu C, Ma X, et al. Growing software: objective, methodology, and technology. IEEE Trans Comput Soc Syst, 2023, 10: 833–842
21 Heineman G T, Councill W T. Component-Based Software Engineering: Putting the Pieces Together. Boston: Addison-Wesley Longman Publishing Co Inc, 2001
22 Karpathy A. Twitter post on hallucination in LLMs. 2023. https://x.com/karpathy/status/1733299213503787018
23 LeCun Y. Mathematical obstacles on the way to human-level AI. AMS Josiah Willard Gibbs Lecture at the 2025 Joint Mathematics Meetings, 2025. https://www.youtube.com/watch?v=ETZfkkv6V7Y
24 Nielsen C B, Larsen P G, Fitzgerald J, et al. Systems of systems engineering. ACM Comput Surv, 2015, 48: 1–41
25 Boardman J, Sauser B. System of systems-the meaning of of. In: Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering, 2006
26 Kroes P, Franssen M, Poel I, et al. Treating socio-technical systems as engineering systems: some conceptual problems. Syst Res, 2006, 23: 803–814
27 Berners-Lee T, Fischetti M, Dertouzos M L. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. New York: HarperInformation, 2000
28 Mumford E. The story of socio-technical design: reflections on its successes, failures and potential. Inf Syst J, 2006, 16: 317–342
29 Cleland-Huang J, Agrawal A, Vierhauser M, et al. Extending MAPE-K to support human-machine teaming. In: Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems. New York: Association for Computing Machinery, 2022. 120–131
30 de Lemos R. Human in the loop: what is the point of no return? In: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. New York: Association for Computing Machinery, 2020. 165–166
31 Kelly K. AR will spark the next big tech platform—call it mirrorworld. Wired, 2019
32 Weyns D. Handbook of Software Engineering. Cham: Springer International Publishing, 2019. 399–443
33 Ma X, Gu T, Song W. Engineering Trustworthy Software Systems. Cham: Springer International Publishing, 2018. 143–175
34 Karpathy A. Software is changing (again). Talk at Y Combinator AI Startup School, 2025
35 Ebert C, Ray R. Test-driven requirements engineering. IEEE Softw, 2021, 38: 16–24
36 Selic B. The pragmatics of model-driven development. IEEE Softw, 2003, 20: 19–25
37 France R B, Rumpe B. Model-driven development of complex software: a research roadmap. In: Proceedings of Future of Software Engineering, 2007. 37–54
38 Naveed H, Arora C, Khalajzadeh H, et al. Model driven engineering for machine learning components: a systematic literature review. Inf Software Tech, 2024, 169: 107423
39 Dings?yr T, Nerur S, Balijepally V G, et al. A decade of agile methodologies: towards explaining agile software development. J Syst Software, 2012, 85: 1213–1221
40 Fitzgerald B, Stol K J. Continuous software engineering: a roadmap and agenda. J Syst Software, 2017, 123: 176–189
41 Leite L, Rocha C, Kon F, et al. A survey of DevOps concepts and challenges. ACM Comput Surv, 2019, 52: 1–35
42 Clarke E M, Wing J M. Formal methods. ACM Comput Surv, 1996, 28: 626–643
43 Woodcock J, Larsen P G, Bicarregui J, et al. Formal methods. ACM Comput Surv, 2009, 41: 1–36
44 Seshia S A, Sadigh D, Sastry S S. Toward verified artificial intelligence. Commun ACM, 2022, 65: 46–55
45 Amershi S, Weld D, Vorvoreanu M, et al. Guidelines for human-AI interaction. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019. 1–13
46 Shneiderman B. Human-centered artificial intelligence: reliable, safe & trustworthy. Int J Hum Comput Interact, 2020, 36: 495–504
47 Fan A, Gokkaya B, Harman M, et al. Large language models for software engineering: survey and open problems. In: Proceedings of the IEEE/ACM International Conference on Software Engineering: Future of Software Engineering, 2023. 31–53
48 Ma Y, Song Z, Zhuang Y, et al. A survey on vision-language-action models for embodied AI. 2024. ArXiv:2405.14093
49 Tao F, Zhang H, Liu A, et al. Digital twin in industry: state-of-the-art. IEEE Trans Ind Inf, 2019, 15: 2405–2415
50 Fuller A, Fan Z, Day C, et al. Digital twin: enabling technologies, challenges and open research. IEEE Access, 2020, 8: 108952
51 Werthner H, Prem E, Lee E A, et al. Perspectives on Digital Humanism. Cham: Springer International Publishing AG, 2021
52 Lessig L. Code and Other Laws of Cyberspace. New York: Basic Books, 1999
53 Akkermans H. Introduction to Digital Humanism. Cham: Springer Nature Switzerland, 2024. 65–81
54 Lee E A. Certainty vs. intelligence. In: Proceedings of Bridging the Gap Between AI and Reality: Second International Conference, AISoLA 2024. Berlin: Springer-Verlag, 2025. 20–32
55 Shavit Y, Agarwal S, Brundage M, et al. Practices for Governing Agentic AI Systems. White Paper OpenAI Publication, 2025
56 Google Cloud. What is agentic AI? Definition and differentiators. 2026
57 IBM. What is agentic AI? 2026
58 OWASP GenAI Security Project. Agentic AI—Threats and Mitigations. Whitepaper, OWASP Foundation, 2025
59 Sifakis J, Li D, Huang H, et al. A reference architecture for autonomous networks: an agent-based approach. 2025. ArXiv:2503.12871
60 Fung P, Bachrach Y, Celikyilmaz A, et al. Embodied AI agents: modeling the world. 2025. ArXiv:2506.22355
61 Wang H, Poskitt C M, Sun J. Agentspec: customizable runtime enforcement for safe and reliable LLM agents. 2025. ArXiv:2503.18666
62 Li Y, Chen Y, Wen H, et al. Vigil: runtime enforcement of behavioral specifications in AI agent skills. 2026. ArXiv:2606.26524
63 Mei H, Guo Y. Toward ubiquitous operating systems: a software-defined perspective. Computer, 2018, 51: 50–56
64 Lin Q, Zhang H, Lou J, et al. Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering, 2016. 102–111
65 Jin X, An J, Zhan B, et al. Inferring switched nonlinear dynamical systems. Form Asp Comput, 2021, 33: 385–406
66 Wang Y, Zhan S S, Wang Z, et al. Joint differentiable optimization and verification for certified reinforcement learning. In: Proceedings of the 14th ACM/IEEE International Conference on Cyber Physical Systems, 2023. 132–141
67 Pham L, Ha H, Zhang H. BARO: robust root cause analysis for microservices via multivariate Bayesian online change point detection. Proc ACM Softw Eng, 2024, 1: 2214–2237
68 Hoenicke J, Olderog E. CSP-OZ-DC: a combination of specification techniques for processes, data and time. Nord J Comput, 2002, 9: 301–334
69 Neema S, Parikh R, Jagannathan S. Building resource adaptive software systems. IEEE Softw, 2019, 36: 103–109
70 Shen W, Han M, Liu J, et al. Xsched: preemptive scheduling for diverse xpus. In: Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation, 2025. 671–692

71 Craun M, Oswald A, Williams D. Enabling eBPF on embedded systems through decoupled verification. In: Proceedings of the 1st Workshop on eBPF and Kernel Extensions, 2023. 63–69
72 Kaffes K, Humphries J T, Mazières D, et al. Syrup: user-defined scheduling across the stack. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, 2021. 605–620
73 Cao D G, Xue D L, Ma Z Y, et al. XiUOS: an open-source ubiquitous operating system for industrial Internet of Things. Sci China Inf Sci, 2022, 65: 117101
74 Ginzburg S, Shahrad M, Freedman M J. Vectorvisor: a binary translation scheme for throughput-oriented GPU acceleration. In: Proceedings of the 2023 USENIX Annual Technical Conference, 2023. 1017–1037
75 Ramesh A, Huang T, Titzer B L, et al. Empowering webassembly with thin kernel interfaces. In: Proceedings of the 20th European Conference on Computer Systems, 2025. 1–20
76 Dao T, Fu D, Ermon S, et al. Flashattention: fast and memory-efficient exact attention with io-awareness. Adv Neural Inf Process Syst, 2022, 35: 16344–16359
77 Dao T. Flashattention-2: faster attention with better parallelism and work partitioning. 2023. ArXiv:2307.08691
78 Aminabadi R Y, Rajbhandari S, Awan A A, et al. Deepspeed-Inference: enabling efficient inference of transformer models at unprecedented scale. In: Proceedings of SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, 2022. 1–15
79 Pope R, Douglas S, Chowdhery A, et al. Efficiently scaling transformer inference. In: Proceedings of the Conference on Machine Learning and Systems, 2023
80 Yu G I, Jeong J S, Kim G W, et al. Orca: a distributed serving system for transformer-based generative models. In: Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation, 2022. 521–538
81 Chen L, Ye Z, Wu Y, et al. Punica: multi-tenant LoRA serving. 2023. ArXiv:2310.18547
82 Sheng Y, Zheng L, Yuan B, et al. Flexgen: high-throughput generative inference of large language models with a single GPU. In: Proceedings of International Conference on Machine Learning, 2023. 31094–31116
83 Wu B, Zhong Y, Zhang Z, et al. Fast distributed inference serving for large language models. 2023. ArXiv:2305.05920
84 Agrawal A, Panwar A, Mohan J, et al. Sarathi: efficient LLM inference by piggybacking decodes with chunked prefills. 2023. ArXiv:2308.16369
85 Patel P, Choukse E, Zhang C, et al. Splitwise: efficient generative LLM inference using phase splitting. 2023. ArXiv:2311.18677
86 Zhong Y, Liu S, Chen J, et al. DistServe: disaggregating prefill and decoding for goodput-optimized large language model serving. In: Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation, 2024. 193–210
87 Microsoft. Retrace your steps with Recall. 2024. https://support.microsoft.com/en-us/windows/retrace-your-steps-with-recall-aa03f8a0-a78b-4b3e-b0a1-2eb8ac48701c
88 Yang J, Jin H, Tang R, et al. Harnessing the power of LLMs in practice: a survey on ChatGPT and beyond. 2023. ArXiv:2304.13712
89 Li Y, Wen H, Wang W, et al. Personal LLM agents: insights and survey about the capability, efficiency and security. 2024. ArXiv:2401.05459
90 Zhang Z, He B, Zhang Z. Practical edge kernels for integer-only vision transformers under post-training quantization. In: Proceedings of the Conference on Machine Learning and Systems, 2023
91 Xu D, Zhang H, Yang L, et al. Empowering 1000 tokens/second on-device LLM prefilling with MLLM-NPU. 2024. ArXiv:2407.05858
92 Yang B, He L, Ling N, et al. EdgeFM: leveraging foundation model for open-set learning on the edge. 2023. ArXiv:2311.10986
93 Yi R, Guo L, Wei S, et al. Edgemoe: fast on-device inference of moe-based large language models. 2023. ArXiv:2308.14352
94 Kong R, Li Y, Feng Q, et al. Serving MoE models on resource-constrained edge devices via dynamic expert swapping. 2023. ArXiv:2308.15030
95 Xu D, Zhang H, Yang L, et al. Fast on-device LLM inference with NPUs. 2024. ArXiv:2407.05858
96 Song Y, Mi Z, Xie H, et al. Powerinfer: fast large language model serving with a consumer-grade GPU. 2023. ArXiv:2312.12456
97 Guo L, Choe W, Lin F X. Sti: Turbocharge NLP inference at the edge via elastic pipelining. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2023. 791–803
98 Alizadeh K, Mirzadeh I, Belenko D, et al. LLM in a flash: efficient large language model inference with limited memory. 2023. ArXiv:2312.11514
99 Bader J, Scott A, Pradel M, et al. Getafix: learning to fix bugs automatically. Proc ACM Program Lang, 2019, 3: 1–27
100 Zhang J, Renganarayana L, Zhang X, et al. Encore: exploiting system environment and correlation information for misconfiguration detection. In: Proceedings of Architectural Support for Programming Languages and Operating Systems, 2014. 687–700
101 Liu X, Li H, Lu X, et al. Understanding diverse usage patterns from large-scale appstore-service profiles. IEEE Trans Software Eng, 2018, 44: 384–411
102 Zhang Y, Huang Y. "Learned": operating systems. SIGOPS Oper Syst Rev, 2019, 53: 40–45
103 Laga A, Boukhobza J, Koskas M, et al. Lynx: a learning Linux prefetching mechanism for SSD performance model. In: Proceedings of the 5th Non-Volatile Memory Systems and Applications Symposium, 2016. 1–6
104 Birkner R, Drachsler-Cohen D, Vanbever L, et al. Config2spec: mining network specifications from network configurations. In: Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation, Santa Clara, 2020. 969–984
105 Ma X, Wang Y, Yao Y, et al. Caution for the environment: multimodal LLM agents are susceptible to environmental distractions. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, 2025. 22324–22339
106 Packer C, Wooders S, Lin K, et al. MemGPT: towards LLMs as operating systems. 2023. ArXiv:2310.08560
107 Lu T, Zheng H, Zhang T, et al. Interactout: leveraging interaction proxies as input manipulation strategies for reducing smartphone overuse. In: Proceedings of the 2024 ACM Conference on Human Factors in Computing Systems, 2024. 1–19
108 Damm W, Hess D, Schweda M, et al. A reference architecture of human cyber-physical systems-part I: fundamental concepts. ACM Trans Cyber Phys Syst, 2024, 8: 1–32
109 MathWorks Inc. Simulink User's Guide. 2013. http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf
110 MathWorks Inc. Stateflow User's Guide. 2013. http://www.mathworks.com/help/pdf_doc/stateflow/sf_ug.pdf
111 Object Management Group. OMG Systems Modeling Language Version 1.6 Beta Specification. 2019. http://www.omg.org/spec/SysML
112 Tiller M. Introduction to Physical Modeling with Modelica. Boston: Springer, 2001
113 Henzinger T A. The theory of hybrid automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, 1996. 278–292
114 He J. A Classical Mind: Essays in Honour of Charles Antony Richard Hoare. Hemel Hempstead: Prentice Hall International Ltd., 1994. 171–189
115 Xu X, Talpin J P, Wang S, et al. HPC: a calculus for hybrid and mobile systems. Proc ACM Program Lang, 2025, 9: 1158–1183
116 OpenAI. ChatGPT-Introducing ChatGPT, 2022. https://openai.com/index/chatgpt
117 Albrecht S V, Christianos F, Schäfer L. Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. Cambridge: MIT Press, 2024
118 Hoare C A R, He J. Unifying Theories of Programming. Englewood Cliffs: Prentice Hall, 1998
119 Xu X, Talpin J, Wang S, et al. Semantics foundation for cyber-physical systems using higher-order UTP. ACM Trans Softw Eng Methodol, 2023, 32: 1–48
120 Bengler K, Damm W, Lüdtke A, et al. A references architecture for human cyber physical systems, Part II: fundamental design principles for human-CPS interaction. ACM Trans Cyber Phys Syst, 2024, 8: 1–27

121 Macenski S, Foote T, Gerkey B, et al. Robot operating system 2: design, architecture, and uses in the wild. Sci Robot, 2022, 7: 6074
122 Somers R J, Douthwaite J A, Wagg D J, et al. Digital-twin-based testing for cyber-physical systems: a systematic literature review. Inf Softw Technol, 2023, 156: 107145
123 Fang Y, Jin Z, An J, et al. Enhancing transformation from natural language to signal temporal logic using LLMs with diverse external knowledge. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, 2025. 10446–10458
124 Fang Y, Jin Z, An J, et al. RESTL: reinforcement learning guided by multi-aspect rewards for signal temporal logic transformation. In: Proceedings of the 40th Conference of the Association for the Advancement of Artificial Intelligence, the 38th Conference on Innovative Applications of Artificial Intelligence, the 16th Symposium on Educational Advances in Artificial Intelligence, 2026. 30682–30689
125 Weyns D, Bäck T, Vidal R, et al. The vision of self-evolving computing systems. J Integr Des Process Sci, 2022, 26: 4–26
126 Li J, Zhang M, Li N, et al. Generative AI for self-adaptive systems: state of the art and research roadmap. ACM Trans Auton Adapt Syst, 2024, 19: 1–60
127 Marron A, Harel D. Early fault-detection in the development of exceedingly complex reactive systems. In: Proceedings of the 13th International Conference on Model-Based Software and Systems Engineering, 2025. 321–329
128 Damm W, Fränzle M, Kerscher A J, et al. A reference architecture of human cyber-physical systems-part III: semantic foundations. ACM Trans Cyber Phys Syst, 2024, 8: 1–23
129 Cai Y, Hou Z, Sanan D, et al. Automated program refinement: guide and verify code large language model with refinement calculus. Proc ACM Program Lang, 2025, 9: 2057–2089
130 Zhang Y, Cai Y, Zuo X, et al. Position: trustworthy AI agents require the integration of large language models and formal methods. In: Proceedings of the 42nd International Conference on Machine Learning, 2025
131 Cao J, Lu Y, Li M, et al. From informal to formal-incorporating and evaluating LLMs on natural language requirements to verifiable formal proofs. In: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, 2025. 26984–27003
132 Zhang Y, Emma S Y, En A L J, et al. RvLLM: LLM runtime verification with domain knowledge. In: Proceedings of the 39th Annual Conference on Neural Information Processing Systems, 2025
133 Jin W, Cao Y, Su J, et al. ALMGuard: safety shortcuts and where to find them as guardrails for audio language models. In: Proceedings of the 39th Annual Conference on Neural Information Processing Systems, 2025
134 Dai W, Li B, Dong N, et al. Fracface: breaking the visual clues fractal-based privacy-preserving face recognition. In: Proceedings of the 39th Annual Conference on Neural Information Processing Systems, 2025
135 An J, Chen M, Zhan B, et al. Learning one-clock timed automata. In: Proceedings of the 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2020. 444–462
136 Yu H, Ma B, Chen M, et al. Derivative-agnostic inference of nonlinear hybrid systems. 2025
137 Yu S, Wang T, Wang J. Loop invariant inference through SMT solving enhanced reinforcement learning. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, 2023. 175–187
138 Zhao H, Zeng X, Chen T, et al. Learning safe neural network controllers with barrier certificates. Form Asp Comput, 2021, 33: 437–455
139 Liu H, Sun J, Li Z, et al. Proofaug: efficient neural theorem proving via fine-grained proof structure analysis. In: Proceedings of the 42nd International Conference on Machine Learning, 2025
140 Wang H, Xin H, Zheng C, et al. Lego-prover: neural theorem proving with growing libraries. In: Proceedings of the 12th International Conference on Learning Representations, 2024
141 Ye Z, Ji R, Xiong Y, et al. Accelerating syntax-guided program synthesis by optimizing domain-specific languages. Proc ACM Program Lang, 2026, 10: 1066–1093
142 Ji R, Kong C, Xiong Y, et al. Improving oracle-guided inductive synthesis by efficient question selection. Proc ACM Program Lang, 2023, 7: 819–847
143 Antinyan V. Revealing the complexity of automotive software. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020. 1525–1528
144 El-Kady A H, Halim S, El-Halwagi M M, et al. Analysis of safety and security challenges and opportunities related to cyber-physical systems. Process Saf Environ Prot, 2023, 173: 384–413
145 Shivaramu P. Development of a cyber-physical system for real-time production monitoring in a digital factory. Available at SSRN 5118240, 2025
146 Acharya S, Khan A A, Päivärinta T. Interoperability levels and challenges of digital twins in cyber-physical systems. J Ind Inf Integr, 2024, 42: 100714
147 Siregar B, Gunawan D, Andayani U, et al. Food delivery system with the utilization of vehicle using geographical information system and a star algorithm. In: Proceedings of Journal of Physics: Conference Series, 2017. 801: 012038
148 Liu Y, Yang C, Jiang L, et al. Intelligent edge computing for IoT-based energy management in smart cities. IEEE Netw, 2019, 33: 111–117
149 Sayeduzzaman M, Hasan T, Nasser A A, et al. Automated Secure Computing for Next-Generation Systems. Hoboken: Wiley, 2024. 243–273