Acta Scientific COMPUTER SCIENCES

Volume 7 Issue 7 October 2025

Review Article

Technological Stages of Neural Network AI Generation of System Program Code Based on Modular Neuro Integration

Evgeny Bryndin*

Research Department, Research Center «Natural Informatics», Novosibirsk, Russia

***Corresponding Author:** Evgeny Bryndin, Research Department, Research Center «Natural Informatics», Novosibirsk, Russia.

Received: August 01, 2025

Published: September 06, 2025

Abstract

A trend of vibe coding has emerged in the world of technology – code generation by neural networks from natural language. For example, Cursor via ChatGPT 4.1 can generate various software modules based on descriptions of their functions in natural language. Creating large software systems from generated modules requires integration. Neurocomplexation of software modules is the process of integrating or combining various software modules to create complex systems based on neural models or artificial neural networks. This approach is proposed to be used in the field of artificial intelligence and machine learning to build complex systems where individual modules interact and jointly perform tasks. Promising areas of application are, firstly, the creation of cognitive systems and intelligent ensembles of agents and assistants. Secondly, modeling of thinking and brain function for research in neuroscience, and thirdly, the development of complex solutions in the field of automation and robotics. The key features of the neural network integration process are, firstly, the integration of modules with different functions (recognition, data processing, training). Secondly, the use of neural network algorithms for adaptation and self-training. Thirdly, ensuring the flexibility and scalability of the system.

Keywords: Neural Network Technology; AI Generation; System Program Code; Modular Neural Integration

Introduction

Currently, neural network system programming is developing. This is a direction of artificial intelligence that combines the development and implementation of neural network models in system software to solve various problems, such as automation, optimization, security and monitoring of systems. This area includes the creation of software solutions that use neural networks to process large amounts of data, make decisions and manage systems at a low level. The main aspects of neural network system programming include:

- Integration of neural networks into system software - development of modules and services that use neural networks to perform tasks, such as anomaly detection, forecasting or automatic control.
- **Training and optimization of models:** Preparation and configuration of neural network models for specific system tasks, which requires knowledge in both machine learning and system programming.
- **Use of specialized frameworks:** Such as TensorFlow, PyTorch, ONNX, which allow you to integrate neural networks into system applications and drivers.
- Working with low-level and high-level programming languages: to develop efficient and fast solutions using neural networks.
- **Ensuring security and reliability:** The implementation of neural network solutions should take into account security requirements, especially in critical systems.

The areas of application are, firstly, automatic control of equipment and robotics, secondly, detection and elimination of failures in systems, thirdly, optimization of resources and energy consumption, fourthly, security and authentication at the level of system services.

In general, neural network system programming requires interdisciplinary knowledge in the field of machine learning, sys-

tem programming and architecture of computing systems [1]. It is based on full integration with AI tools that understand human language and write code for you. Here you only set the direction, simply describing what exactly you want to get, and the AI neural network does the rest and works in the format of a live dialogue. In fact, it is enough to write a high-quality request, and in a few minutes you will receive a ready-made system program. Today, 25% of startups in the latest stream of Y Combinator - a venture fund for startups - use code that is 95% generated by neural networks. The EI Capitan supercomputer, used as a high-performance computing system, and the OpenAI supercomputer-class infrastructure to support the training and deployment of large-scale AI models, are effectively promoting the development of neural network systems programming. According to GitHub, the implementation of Copilot, the OpenAI supercomputer-class infrastructure, and the EI Capitan Supercomputer increases the productivity of programmers so much that it even affects the world's GDP.

The article discusses the technological stages of neural network AI generation of system code based on modular neural integration for the development of neural network system programming.

Creating self-learning neural networks for module code generation

A self-learning neural network for modular code generation is capable of learning, adapting, and generating code based on described modules or their templates. Let's consider the main components and stages of developing such systems.

- Data collection and preparation:
  - Creation of extensive datasets of source code, documentation, comments, and templates.
  - Data markup for training models to understand the modular structure of programs.
- Neural network model:
  - Using transformers (e.g., GPT, Codex) or other architectures that have proven themselves in code generation.
  - Models are trained on examples, breaking down tasks into modules or components, which facilitates modular generation.
- Modularity and structuring:
  - Development of a neural network that solves the problem of modular code generation.
  - Training a neural network to create holistic, coherent programs of modules.
- Self-training and retraining:
  - **Implementation of active learning mechanisms:** The system learns from new data collected from user corrections, feedback or code execution.
  - Using reinforcement learning methods to improve the quality of generation.
- Interactivity and modularity:
  - Providing the user with the ability to add, edit and combine modules so that the neural network automatically ensures their coherence.
- Integration and automation:
  - Implementation of the system into development processes, CI/CD, automated testing and deployment. - Using version control systems and metadata to track changes and improvements.
- Key challenges and solutions:
  - Ensuring the correctness and security of the generated code.
  - Training on a variety of data to increase versatility.
  - Ensuring modular structuring for easy reuse of components.
  - Generating templates and components for web applications, microservices or embedded systems.
  - Automatic generation of API wrappers and integration modules.
  - Tools for automatic addition and correction of code based on modular templates.

Creating self-learning neural networks for modular code generation requires a combination of knowledge in machine learning, system architecture, programming and automation [2]. This allows to significantly speed up the development process, improve its quality and ensure the adaptability of systems to new requirements.

The creators of self-learning neural networks for modular code generation are specialists in AI programming. Intelligent programming specialists are professionals involved in the development of methods, systems and technologies based on artificial intelligence to automate the programming process, improve code quality and increase the efficiency of software development. Their key competencies and areas of activity include:

- Knowledge of artificial intelligence and machine learning methods:
  - Development and implementation of model training algorithms to automate programming tasks.
  - Use of neural network architectures, such as transformers, to generate and analyze code.
- Natural language processing (NLP):
  - Creating systems that understand and interpret requirements and descriptions of tasks in natural language.
  - Developing models that can transform text queries into working code.
- Programming automation:
  - Developing tools for automatic writing, correction and optimization of program code.
  - Implementation of intelligent code completion, refactoring and testing systems.
- Knowledge of programming and software architectures:
  - Deep understanding of programming languages, data structures, algorithms.
  - Skills in designing systems using AI to support development.
- Data engineering and data management:
  - Collecting, preparing and processing large volumes of data for training models.
  - Creating and maintaining datasets with code and requirements.
- Development and implementation of expert systems:
  - Building systems that can make decisions and give recommendations on programming based on knowledge and data.
- **Evaluation and testing of intelligent programming systems:
  - Developing metrics for assessing the quality of the generated code.
  - Conducting testing and validation of automated solutions.
- Interdisciplinary skills:
  - Understanding aspects of linguistics, logic, automata theory and computational theory.
  - Ability to integrate knowledge from various fields to create complex systems.
- Learning and development
  - Continuously monitoring new research and technologies in the field of AI and programming.
  - Participation in scientific and practical developments to improve methods of intelligent programming.
- Communication and Collaboration:
  - Effective interaction with development teams, researchers and business stakeholders to implement solutions. Intelligent programming specialists are experts at the intersection of artificial intelligence and software development, creating intelligent systems that can automate and optimize the programming process, increasing its efficiency and quality.

The creators of neural networks for generating program code from natural language must have a number of competencies, including both technical and interdisciplinary skills. The main ones are:

- Knowledge of machine learning and deep learning:
  - Understanding of neural network architectures (Transformers, GPT, BERT, etc.).
  - Skills in training, setting up and optimizing models.
- Programming and software development:
  - Proficiency in programming languages (Python, Java, C++, etc.).
  - Experience with ML frameworks (TensorFlow, PyTorch).
- Natural language processing (NLP):
  - Knowledge of text preprocessing methods, tokenization, training language models.
  - Understanding of the specifics of the programming language and its syntax.
- Knowledge of programming and development concepts:
  - Knowledge of data structures, algorithms, design patterns.
  - Understanding of requirements for code quality, readability and security.
- Analytical skills and data processing:
  - Ability to collect, clean and analyze datasets of program code.
  - Creation and management of training data corpora.
- Skills for evaluating and testing models:
  - Methods for assessing the quality of code generation.
  - Development of tests and automated systems for checking the generated code.
- Knowledge of the subject area:
  - Understanding of the context of application and requirements for software.
- Interdisciplinary skills:
  - Ability to work at the intersection of programming, linguistics and machine learning.
- Communication and teamwork skills:
  - Explaining technical solutions, interacting with a team of developers and researchers.
- Continuous learning and research:
  - Keep up to date with current research, new architectures and methods in AI and NLP for code. These competencies enable the creation of effective neural networks capable of generating high-quality and secure software code based on natural language requests.

Description of the system by module promts in natural language

Description of a system using modular prompts in natural language is based on the use of modular prompts, which allow for flexible and efficient management of the user interaction process and task execution [3]. Each module is a separate prompt responsible for a specific function or aspect of the dialogue. System structure:

- **General user input:** The user enters a message or question to be processed.
- **Pre-processing module:** Analyzes the input data, determines the context, and highlights key parameters.
- **Task segmentation module:** Divides the task into subtasks or parts to determine which prompts to use for each stage.
- **Prompt selection module:** Based on the analysis, selects appropriate modular prompts to generate a response. For example:
    - Prompt for clarifying information
    - Prompt for providing information
    - Prompt for clarifying requirements
- **Response generation module:** Uses the selected prompts to form a relevant and coherent response.
- **Post-processing module:** Checks and corrects the generated response if necessary to ensure quality and accuracy.
- System advantages:
    - **Flexibility:** Individual prompts can be easily added or changed without having to rewrite the entire system.
    - **Modularity:** Provides ease of customization and scalability.
    - **Quality control**: Each part of processing and generation can be optimized separately.

Neural network generation of software modules of the system by promts

Neural network generation of software modules of a system from prompts is an approach in which artificial intelligence based on deep learning models automatically creates software components of a system following given instructions or descriptions (prompts) in natural language [4-7]. This method allows to speed up development, increase flexibility and simplify the creation of complex systems without the need for manual programming of each module.

The main ideas and stages of this approach:

- **Using prompts in natural language:** The user or developer specifies a description of the required module - its functionality, interface, behavior features - in the form of a text prompt, for example, creating an image processing module that recognizes faces and highlights key features.
- **Training and application of neural networks:** Models such as GPT, Codex, or specialized systems for code generation are trained on large volumes of software data and can interpret prompts and transform them into source code or architectural diagrams.
- **Automatic code generation:** Based on the received prompt, the neural network creates a software module - functions, classes, interfaces, or even entire system components. This may include writing code in programming languages, creating configuration files, or describing an API.
- **Integration and testing:** Generated modules are linked into the system and automatic tests are performed to check their correctness and compliance with requirements.
- **Iterative improvement:** If necessary, the user can refine the prompts or set new instructions, and the neural network will refine or create new versions of the modules.

The advantages of this approach are, firstly, rapid development: significantly reduces the time for creating software components, secondly, flexibility and adaptability, it is easy to change requirements by simply changing the prompts, thirdly, accessibility for non-specialists: it allows people without deep knowledge of programming to create complex systems, fourthly, it is possible to automatically generate many modules for different tasks. For example, the user asks the prompt: “Create a module for parsing JSON responses and extracting user data. The neural network generates source code in Python or another language that implements the parser and data extraction function, which is then integrated into the system.

Creation of self-learning neural networks for integration of software modules

Self-learning neural networks for software module assembly allow you to automate and optimize the processes of selecting and integrating system software modules. Below are the main steps and recommendations for implementing such integration:

- Requirements and goals analysis:
  - **Define the tasks:** Automatic module selection, their configuration, compatibility and optimization.
  - **Specify the performance criteria:** Selection accuracy, speed, adaptability.
- Data collection and preparation:
  - **Collect data on existing modules:** Characteristics, interfaces, dependencies, versions.
  - Collect historical data on assembly, successful and unsuccessful configurations.
  - **Process the data:** Cleaning, normalization, annotation.
- Designing the neural network architecture:
  - **Select the model type:** For example, transformers, recurrent networks or multilayer perceptrons. - You can use models capable of processing structured data and relationships, such as graph neural networks (GNN).
  - Use module characteristics, system requirements and constraints as input data.
- Model training:
  - Divide the data into training, validation, and test sets.
  - Use reinforcement learning or supervised learning, depending on the available data.
  - **Implement self-learning mechanisms:** The model should be able to adjust its parameters based on new data and feedback.
- Implementation of the recommendation system:
  - Build a mechanism that will suggest optimal configurations based on the input requirements.
  - Enable the ability to interact with the user to clarify the requirements.
- Integration and testing:
  - Implement the system in a production environment.
  - Conduct testing on real scenarios, collect feedback for further training.
- Continuous self-learning and updating:
  - Ensure that new data is collected on the system's operation.
  - Tune the model on new data to improve accuracy and adaptability.
  - Use online training or re-training methods.
- Additional recommendations:
  - **Use modularity:** Divide the system into components for requirements analysis, configuration generation, and training.
  - Ensure transparency of model decisions for manual adjustment and trust.
  - Implement mechanisms for checking compatibility and testing the proposed composition of modules.

This is a general approach that can be adapted to the specific tasks and conditions of your system. Technical details on implementation are needed, for example, the choice of architecture or training algorithms.

Neural network integration of modules into a system program

The assembly of modules into a system program automatically collects and configures software modules taking into account the requirements, compatibility and optimization of integration, firstly, automatic linking of modules, secondly, ensuring compatibility and optimization of system configurations. thirdly, self-training of the model based on new data and feedback. Data is prepared, firstly, module characteristics: functions, interfaces, dependencies, versions, resources, secondly, data on collected configurations, successful and unsuccessful examples. Thirdly, metadata on compatibility and limitations, fourthly, preliminary processing: cleaning, normalization, creation of features. The architecture of the neural network model is selected. The model is trained and the integration system is implemented to assemble software modules with the architecture, selection of algorithms or specific tools. Regular retraining of the model is carried out to improve accuracy, adaptability and ensure transparency of solutions.

The requirements for compatibility and optimization of the neural network software module complex include the following main aspects:

- Compatibility of software platforms and environments:
  - Support for major operating systems (Windows, Linux, macOS).
  - Compatibility with popular frameworks for developing neural networks (TensorFlow, PyTorch, Keras, MXNet, etc.).
  - Support for the necessary libraries and dependencies (CUDA, cuDNN for GPU acceleration, OpenCL).
- Hardware compatibility:
  - Adaptation for various types of processors (CPU, GPU, TPU).
  - Ability to work on various devices (local machines, servers, cloud platforms).
- Data and interface standards:
  - Compliance with data format standards (e.g. JSON, Protocol Buffers, ONNX).
  - Ensuring compatibility with external systems and APIs.

- Performance and optimization:
  - Using computation acceleration methods (hardware acceleration, computation graph optimization).
  - Minimizing training and inference time.
  - Efficient memory and resource management.
  - Using quantization, pruning and other modeling techniques to reduce the size of models and increase speed.
- Scalability and modularity:
  - The ability to expand the complex by adding new modules.
  - Ensuring joint operation of modules within a single system.
- Security and reliability:
  - Ensuring data protection and correct operation under various conditions.
  - Error handling and logging.
- Standardization and documentation:
  - Clear description of interfaces and requirements for modules.
  - Support for versioning and compatibility between versions.

Effective implementation of these requirements ensures stable operation of the neural network system programming complex, its scalability and the ability to integrate into various information systems.

Conclusion

Neural network system programming combines the principles of system software development with artificial intelligence and neural network technologies. The main goals are, firstly, the creation of efficient, scalable and optimized systems capable of performing automation tasks, data analysis, resource management and interaction with hardware using neural network models, secondly, the development of neural network algorithm modules integrated into system software (drivers, operating systems, system services). Thirdly, the optimization of computing processes to accelerate inference and training of neural networks at the system level, including the use of hardware accelerators (GPU, FPGA, TPU). Fourthly, ensuring the interaction of neural network models with low-level system components, such as device drivers, operating system kernels. Fifthly, the creation of API interfaces and protocols for the integration of neural network solutions into existing system platforms. Sixthly, working with data streams, memory and resource management to ensure high performance and reliability. Seventh, automated security management of intelligent devices, robotics, monitoring and diagnostic systems, as well as systems capable of self-learning and adaptation in real time.

In general, neural network system programming requires deep knowledge in the field of system programming and computer architecture, as well as in the field of machine learning and neural network technologies, which makes it an important and promising area for the development of modern AGI intelligent information technologies [8-12]. The future of neural network system programming looks very promising and promising. Here are the main directions and trends that can be expected:

- Integration with traditional development tools:
  - Neural networks will become an integral part of standard development environments, helping to automate routine tasks such as coding, refactoring, and testing.
- Automation of system creation:
  - The ability to automatically generate and optimize system components based on high-level requirements.
  - Using neural networks for rapid prototyping and adapting systems to changing conditions.
- Training on large amounts of system data:
  - Neural networks will be trained on huge datasets of logs, metrics, and source code to identify patterns and errors.
  - This will improve the quality and security of system solutions.
- Explainability and control:
  - Developing methods for explaining neural network decisions, which is especially important for system programming, where reliability and transparency are needed.
- New programming paradigms:
  - The transition to more declarative models based on the description of the desired system behavior, rather than specific algorithms.
  - Neural networks will help in the interpretation and implementation of such descriptions.
- Security and reliability:
  - Creation of systems with built-in mechanisms for detecting and correcting errors, as well as protection against threats.
- Evolution of tools:
  - The emergence of new IDEs and platforms specifically designed for working with neural network systems, which will offer automatic recommendations and assistance in development.

- Human-neural network interaction:
  - The development of interfaces that allow specialists to effectively interact with neural networks to create, configure and control system solutions.

Overall, neural network system programming will contribute to the creation of smarter, automated and reliable systems, which will open new horizons in the field of software development and systems engineering.

Bibliography

1. Alexander Kirichenko. "Neural Network Programming. Neurocomputing Toolkit". Created in the Intelligent Publishing System Ridero (2020).

2. Evgeny Bryndin. "Self-learning AI in Educational Research and Other Fields". Research on Intelligent Manufacturing and Assembly 3.1 (2025): 129-137.

3. Andy Smith. "Mastering Prompts: The Art of Interacting with AI". Samizdat (2024): 63.

4. H Peter Alesso. "Vibe Coding by Example". Independently published (2025): 329.

5. Greg Lim. "Vibe Coding for Beginners with Python and ChatGPT". Independently published (2025).

6. David Gillette. "Vibe Coding in Python: The Python Programmers Guide to AI-Powered Programming (Generative AI Mastery)". Independently published (2025): 171.

7. Addy Osmani. "Vibe Coding The Future of Programming: Leveraging Your Experience in the Age of AI-Assisted Coding (First Early Release)". O'Reilly Media, Inc. (2025): 250.

8. Evgeny Bryndin. "Creation of Multi-purpose Intelligent Multimodal Self-Organizing Safe Robotic Ensembles Agents with AGI and cognitive control". *COJ Robotics and Artificial Intelligence (COJRA)* 3.5 (2024): 1-10.

9. Evgeny Bryndin. "Creation of Multimodal Digital Twins with Reflexive AGI Multilogic and Multisensory". *Research on Intelligent Manufacturing and Assembly* 2.1 (2024): 85-93.

10. Evgeny Bryndin. "Formation of Motivated Adaptive Artificial Intelligence for Digital Generation of Information and Technological Actions". *Research on Intelligent Manufacturing and Assembly* 4.1. (2025): 192-199.

11. Evgeny Bryndin. Network Training by Generative AI Assistant of Personal Adaptive Ethical Semantic and Active Ontology". *International Journal of Intelligent Information Systems* 14.2 (2025): 20-25.

12. Addy Osmani. "Beyond Vibe Coding". Publisher(s): O'Reilly Media, Inc. (2025).