CASC Newsletter

Volume 18 | September 2026

IN THIS ISSUE:

From the Director

Contact: Kathryn Mohror

I'm excited to share this edition of the CASC Newsletter, which highlights the creativity and impact of CASC research across a wide range of computing challenges. The first article describes the Shapeshifter project, which is developing decentralized software that allows grid-edge devices to coordinate and form resilient microgrids after disruptions, supported by the open-source Skywing platform and a new Raspberry Pi testbed. Next, we highlight YGM, an asynchronous communication library designed for irregular, distributed workloads, inspired by the needs of distributed graph algorithms. The third article describes important progress toward making artificial intelligence (AI) more interpretable, steerable, and trustworthy for mission-critical science. Finally, CompilerGPT demonstrates how large language models (LLMs) can turn difficult compiler optimization reports into actionable code improvements through iterative compile, optimize, and evaluate workflows.

These accomplishments reflect not only strong research ideas, but also the sustained effort required to turn those ideas into practical algorithms, open-source software, and tools that others can build upon. I'm incredibly proud of the technical excellence,

ingenuity, and collaborative spirit reflected in these accomplishments, and I hope you enjoy learning more about the outstanding work happening across CASC.

Collaborations | Shapeshifter Builds Decentralized Software for a More Resilient Grid

Contact: Alyson Fox

As power systems become more distributed, they also become harder to control during disruptions. Severe weather, cyber-attacks, and physical damage can break connections across the grid, separating critical loads from the generation and control systems they depend on. Traditional approaches often rely on centralized control, which can become a single point of failure when the system is under stress. The Shapeshifter LDRD project is addressing that challenge by developing decentralized software that lets parts of the grid reconfigure into functioning microgrids after hazards. Previously led by CASC researcher Alyson Fox, and currently led by GS-CAD researcher Anna Mauro, the project brings together expertise in applied mathematics, resilient algorithms, and software development to help critical infrastructure adapt instead of fail.

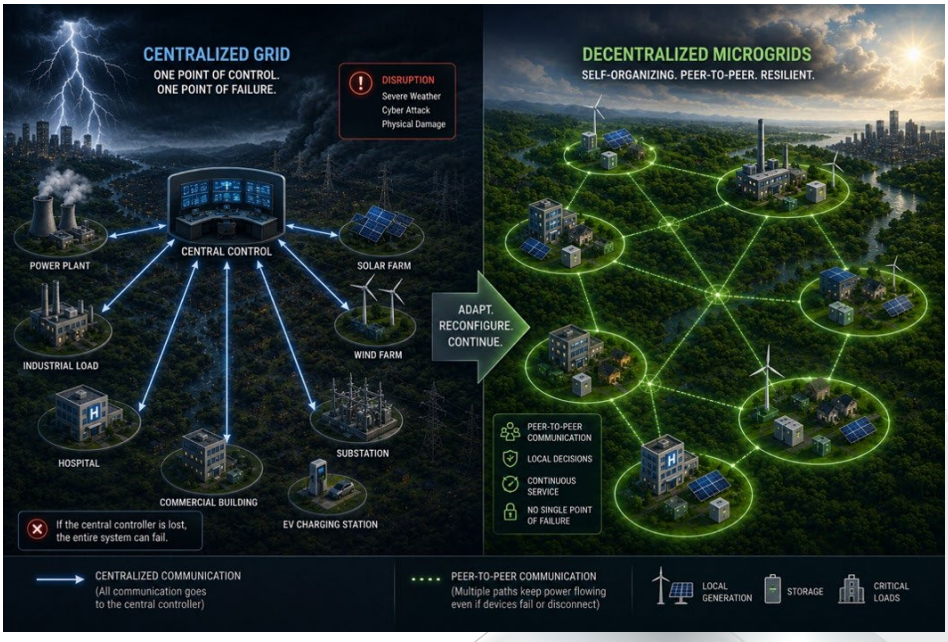

Figure 1: Conceptual evolution of grid control architectures. Traditional power grids rely on centralized control systems, where coordination and decision making flow through a single control point. Emerging approaches such as DynaGrid and PDNM introduce greater flexibility, while Shapeshifter explores decentralized coordination methods designed to support resilient, adaptive grid operation under disrupted conditions.

At the center of the effort is Skywing, an open-source software platform that enables decentralized, asynchronous computation in unreliable environments. Rather than depending on a central coordinator, Skywing allows many devices to work together using local information and peer-to-peer communication. That makes it well suited for the disrupted conditions Shapeshifter is designed to address. A major milestone for the project is the June release of a Python refactor of Skywing. The refactor adds new algorithms and features intended to support future development of decentralized methods. The refactor includes implementations of established decentralized methods such as consensus, while adapting other approaches, including Alternating Direction of Multipliers Methods (ADMM) and Stochastic Gradient Descent, for decentralized settings. It also brings in resilient methods developed in a prior LDRD project, including s-step Asynchronous Conjugate Directions (s-ACD) and a resilient version of Asynchronous Jacobi.

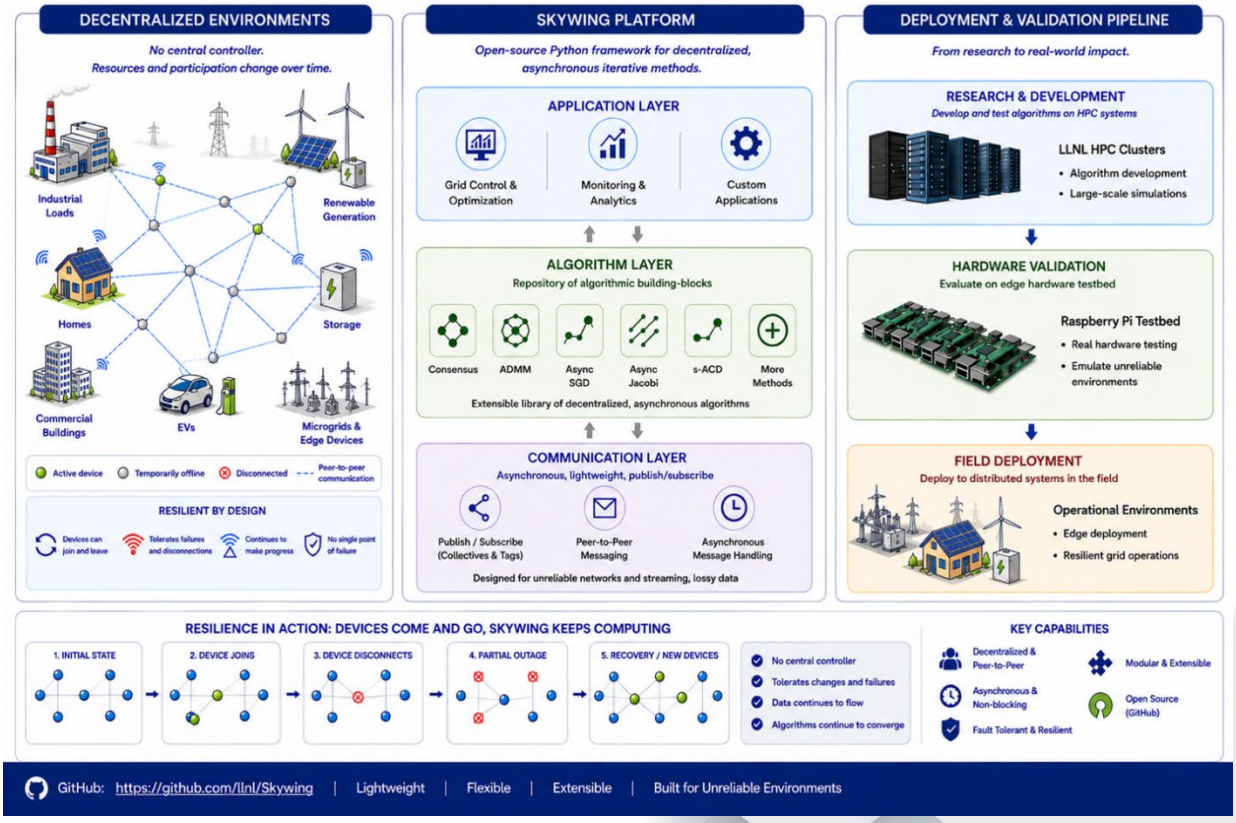

Figure 2: Skywing is an open-source software platform designed for decentralized, asynchronous computing in unreliable environments. The platform enables edge devices to coordinate through peer-to-peer communication without relying on a central controller, allowing computation to continue even as devices join, disconnect, or experience failures. Skywing supports a layered architecture spanning communication, algorithmic building blocks, and application development, with deployment pathways ranging from high-performance computing (HPC) research environments to Raspberry Pi hardware testbeds and real-world field systems.

This software foundation supports Shapeshifter's broader goal of enabling grid-edge devices to self-organize after a disruption and continue supporting critical loads. Instead of assuming that a central controller will always remain available, the project explores how decentralized devices can coordinate locally, form stable control collectives, and help maintain service across fragmented parts of the grid. In this way, the work addresses all three core resilience needs: keeping critical loads online, enabling decentralized control, and reducing single points of failure. The team has also developed a Raspberry Pi testbed that supports Shapeshifter research and provides a flexible environment for future experimentation. In addition to demonstrating project concepts on real hardware, the testbed is already drawing interest from other efforts that want to build on the same decentralized software approach.

Interest in the work is also growing beyond the project itself. After presenting at the Institute for Computational and Experimental Research in Mathematics' 2026 workshop on Asynchronous Methods for Numerical Linear Algebra, the team connected with new users interested in asynchronous methods, an area that can be difficult to demonstrate in conventional tools such as MPI (message passing interface). The project also supports transition work through an effort by the DOE's Office of Cybersecurity, Energy Security, and Emergency Response, taking LDRD-developed ideas and applying them directly to power grid resilience use cases.

Together, these advances position Shapeshifter as both a research effort and a practical software foundation for future resilient infrastructure applications. By combining algorithm development, usable software, and hardware test environments, the project is helping move decentralized grid resilience from concept toward real-world use.

Lab Impact | Asynchronous Computing with YGM

Contact: Roger Pearce and Trevor Steil

Supercomputers are tasked with solving a wide variety of problems including physics simulations, data analytics, and large-scale AI training. The varied application spaces can lead to an equally varied set of demands for inter-process communication. In simple grid-based computations, the communication pattern resembles this grid with pairs of clearly defined processes exchanging arrays of floating-point values. Many graph analytics and data science applications feature unstructured communication patterns with processes that need to exchange variable-sized or non-numeric data, such as sparse vectors.

YGM is an open-source communication library primarily developed by CASC researchers Roger Pearce and Trevor Steil. Inspired by the needs seen in distributed graph algorithms, YGM targets applications with unstructured communication patterns. In these settings, a

mismatch exists between the natural communication an application needs and the requirements for high-performance implementations on modern hardware and what is provided by the MPI standard. Notable mismatches include (a) computation models structured around fetching data from memory on a remote process; (b) applications sending large numbers of small messages; and (c) the need to send variable-length non-numeric data, such as sparse vectors, between processes.

When using MPI or underlying remote direct memory access calls, it is natural to conceptually fetch data from a remote process in order to perform the desired local computation. This has an associated latency that is tolerable when performed relatively infrequently, such as the granularity of a time step within a simulation. This cumulative latency becomes intolerable when the granularity of fetches includes every edge being traversed within a graph.

To avoid these frequent stalls in fetching data, YGM is designed around a fire-and-forget async call. Semantically, the async operations send a function to be executed on the remote process along with arguments to pass to that function, with these arguments often taking the form of pointers to data stored on the remote process. These async calls allow developers to send the local state of a computation to another process where the computation can be resumed with the appropriate data once received, avoiding the stalls associated with fetching the data.

In many applications, the natural unit of communication is very small. For a graph traversal, the message being sent can be as small as an integer identifying the next vertex to visit. Sending messages on HPC networks with such small payloads leads to significant overheads from headers attached to each individual message and limits to scalability from finite packet injection rates.

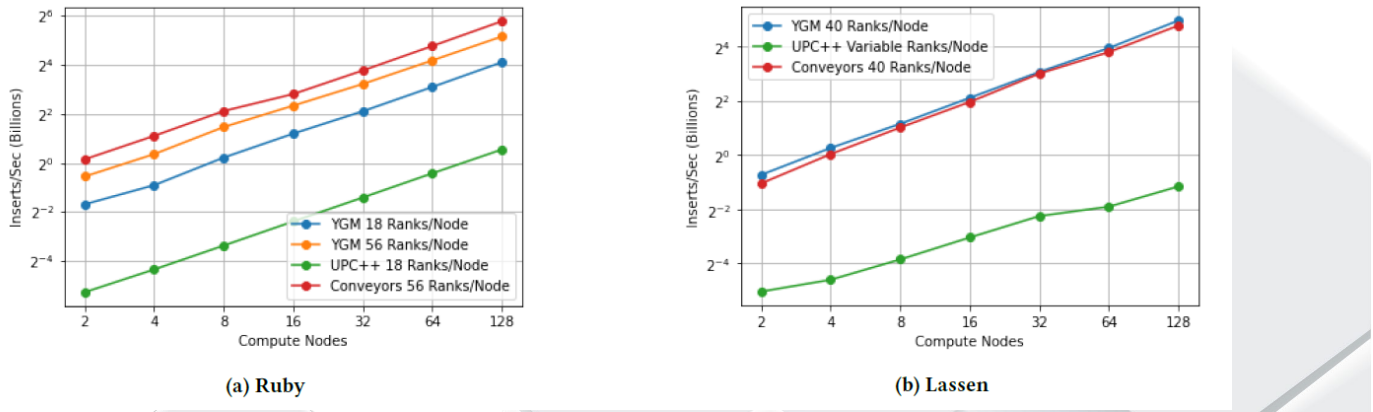

Figure 3: Weak scaling results of insertions into a distributed table using YGM and other distributed frameworks on LLNL's Ruby and Lassen machines [1].

YGM alleviates these issues through its internal message buffering that enables larger bundles of messages to reach the physical network. As applications scale up to larger allocations of nodes, message buffering with a fixed buffer capacity on a process will yield smaller average bundle sizes due to an increase in the number of destination processes. To improve the scalability of message buffering, YGM implements message routing schemes that aggregate messages within compute nodes, maintaining large message sizes sent between compute nodes. In both message buffering and message routing, YGM explicitly increases the latency of individual messages to improve the overall throughput. This exchange allows developers to express their computation using the natural unit of communication without explicitly considering the buffering required for high performance.

One application using YGM is the SaltAtlas nearest neighbors library, led and developed by CASC researchers Geoff Sanders, Keita Iwabuchi, Trevor Steil, and Min Priest. This effort focuses on creating large-scale nearest neighbor index structures for data from arbitrary metric spaces such as Euclidean data using the 2-norm or string data using metrics based on edit distance. SaltAtlas' Distributed Nearest Neighbor Descent (DNND) builds a nearest neighbor graph from an initially random graph [2]. The nearest neighbor descent process then iteratively checks pairs of neighbors from a given point to see if the two neighbors are close enough to be connected. This process builds a graph with a topology that is dynamic during its construction and requires processes exchanging subsets of their points based on this dynamic topology, which is done using YGM.

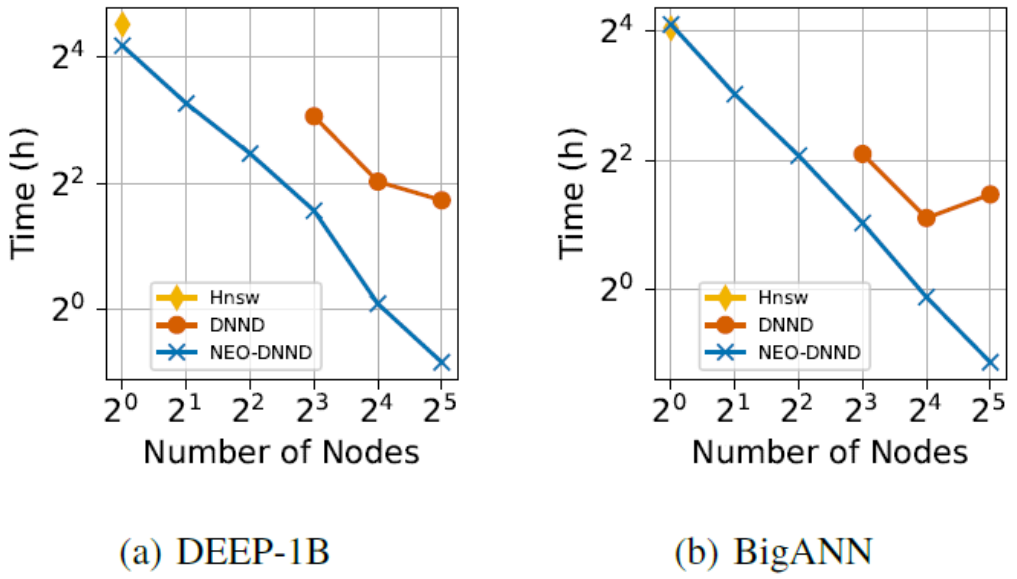

Figure 4: Strong scaling results for building a nearest neighbor index on datasets with 1 billion points using DNND and NEO-DNND [3], a version of DNND optimized for strong-scaling and fixed-width vector data, compared to a multithreaded HNSW (hierarchical navigable small world) index on LLNL's Mammoth cluster. DEEP-1B and BigANN are billion-scale datasets used for benchmarking.

YGM is also being actively used in the Hierarchical Graph-Based Clustering LDRD project, led by Priest and Steil with CASC members Iwabuchi, Grace Li, and Sanders. This project is developing clustering tools for massive datasets using algorithms inspired by the popular HDBSCAN (hierarchical density-based spatial clustering of applications with noise) algorithm. The distributed memory implementation begins by constructing a nearest neighbor graph from the data using SaltAtlas. This nearest neighbor graph is then reduced to an approximate minimum spanning tree using an algorithm designed for YGM's disjoint set data structure. The minimum spanning tree is used to construct a dendrogram from which clusters of points can be extracted. This clustering process has been used to cluster the Observed Antibody Space dataset, which includes 2.1 billion antibody sequences, using a metric similar to a weighted Hamming distance in 2.5 hours on 64 nodes of LLNL's Dane system.

Supercomputer hardware and the MPI standard have been traditionally designed for numerical simulations with well-understood communication patterns. With YGM, projects can now more easily implement algorithms featuring unstructured communication knowing their HPC systems will be used effectively.

[1] T. Steil, *et al*. "Embracing irregular parallelism in HPC with YGM." In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023. doi.org/10.1145/3581784.360710.

[2] K. Iwabuchi, *et al*. "Towards a massive-scale distributed neighborhood graph construction." In *Proceedings of the SC23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023. doi.org/10.1145/3624062.3625132.

[3] K. Iwabuchi, *et al*. "NEO-DNND: communication-optimized distributed nearest neighbor graph construction." In *Proceedings of the SC24-W: Workshops of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2024. doi.org/10.1109/SCW63240.2024.00096.

Advancing the Discipline | Interpretable, Steerable, and Trustworthy AI for Mission-Critical Science

Contact: Kowshik Thopalli

AI is rapidly becoming an indispensable scientific tool. At LLNL, foundation models are beginning to accelerate discovery across biology, materials science, fusion science, and national security. Yet capability alone is not enough, as the Laboratory's mission demands AI systems that are not only accurate but also reliable and trustworthy. Much as commercial aviation earned public confidence through decades of engineering, testing, and continual safety improvements, scientific AI must be accompanied by methods that

allow researchers to understand, validate, and ultimately guide its decisions. Interpretability is therefore an essential ingredient for deploying AI in mission-critical scientific applications.

However, understanding modern foundation models is far from straightforward. Rather than reasoning with concepts that scientists can directly inspect, these models encode knowledge across millions or even billions of numerical features distributed throughout their internal computations. Sparse autoencoders (SAEs) have emerged as one of the leading approaches for translating this hidden representation into human-understandable concepts. An SAE reconstructs a model's internal activations by mapping dense, polysemantic activations onto a larger dictionary of features in which only a handful switch on for any given input. The resulting features tend to become *monosemantic*, with each representing a single human-interpretable concept. SAEs have already found applications in genomics, protein language models, and coding assistants. More importantly, they provide scientists with an interface to foundation models.

Despite this promise, one fundamental question has remained largely unanswered: How good are today's interpretability tools? In a recent Computer Vision and Pattern Recognition conference (CVPR) paper [1], a CASC team (Vivek Narayanaswamy, Shusen Liu, Sam Sakla, Kowshik Thopalli) along with Akshay Kulkarni, Tsui-Wei Weng from the University of California, San Diego, carried out one of the first systematic evaluations of SAEs.

They first addressed a practical obstacle that limited previous studies. Assigning meaningful labels to thousands of discovered concepts typically requires repeated LLM calls for every neuron, making evaluation prohibitively expensive. Instead, the team developed an efficient concept-labeling pipeline that automatically assigns semantic labels at a fraction of the computational cost. Building on this capability, the researchers introduced two inexpensive quantitative metrics that evaluate every discovered concept: *interpretability*, which measures whether a feature represents a coherent semantic concept; and *steerability*, which measures whether manipulating that concept reliably changes the model's behavior.

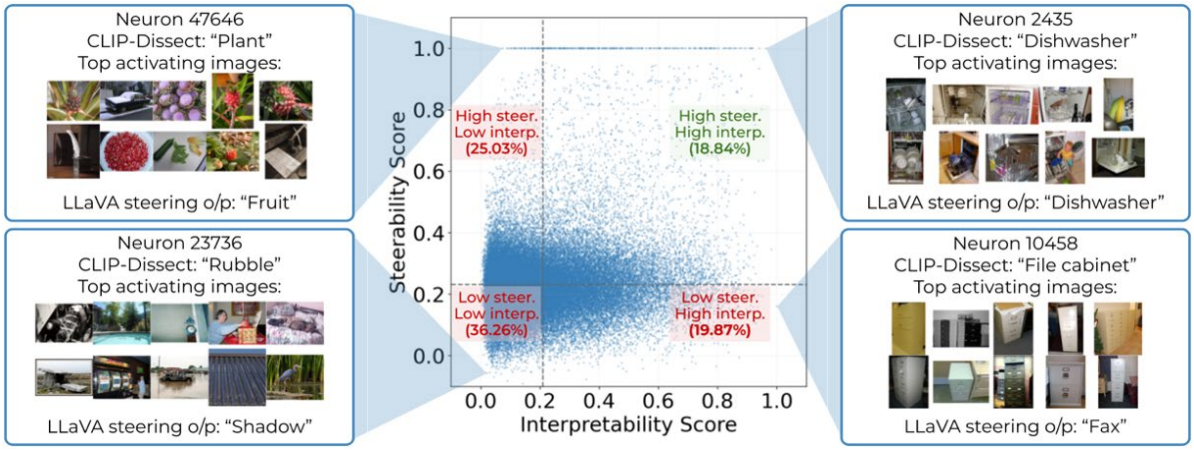

Figure 5: Distribution of neurons according to interpretability and steerability scores for CLIP, a commercial language-image model with qualitative examples. Only the neurons meeting the highest thresholds (green) are retained; the rest are pruned.

The findings revealed substantial room for improvement. Across state-of-the-art vision-language models, including both image captioning and image generation models, only about one in five discovered concepts proved to be both interpretable and useful for steering. Even when the underlying model had learned the necessary knowledge, current interpretability methods often failed to expose it in a form scientists could readily use. This observation motivated a simple question: What if scientists could teach interpretability methods the concepts they already know are important?

Rather than hoping these concepts emerge automatically through unsupervised discovery, the team developed concept bottleneck sparse autoencoders (CB-SAEs). CB-SAEs retain the automatically discovered concepts that are both interpretable and steerable, discard those that are not, and augment the representation with expert-defined concepts supplied by domain scientists. By combining automatic concept discovery with domain expertise, CB-SAEs improved interpretability by 32.1% and steerability by 14.2% while preserving the expressive power of modern foundation models. (Read more about the team's CVPR paper in LLNL Computing news.)

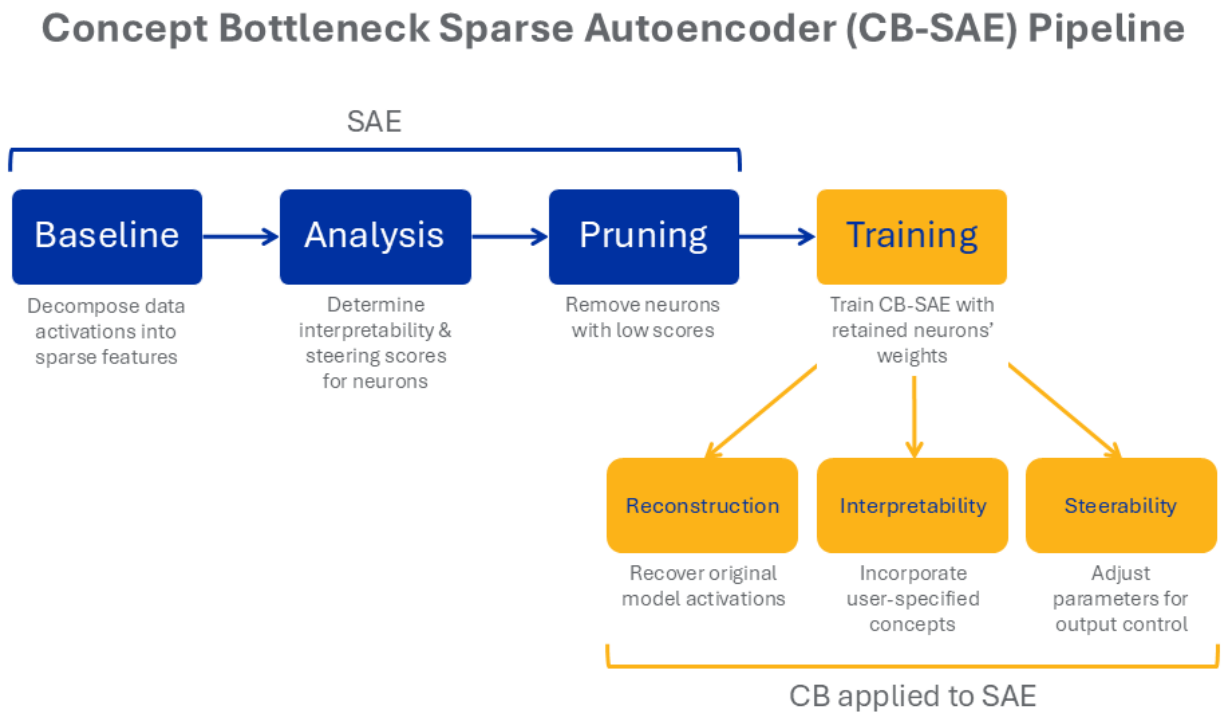

Figure 6: The CB-SAE pipeline begins with identifying and removing SAE neurons that lack sufficient interpretability and steerability. The retained neurons are then augmented with the CB model, which is trained to align with a user-specified, human-understandable concept set. This unified framework was shown to improve interpretability by and steerability across large vision-language models and image-generation tasks.

A natural and important next question is whether these ideas extend beyond vision-language models to scientific foundation models. To explore this, the team collaborated with computational biologists to apply the interpretability and steerability framework to Evo 2, a state-of-the-art genomics foundation model. Preliminary results reveal trends that are broadly consistent with those observed in multimodal foundation models. While current SAE methods recover coarse biological concepts, such as genes, coding sequences, and RNA annotations, they struggle to recover finer grained functional qualifiers that are often more useful for biological interpretation. Likewise, only a small fraction of SAE features achieves high scores on both interpretability and steerability. These encouraging results demonstrate that the evaluation framework developed in the CVPR paper naturally extends to scientific foundation models. The team is now extending CB-SAEs to Evo 2 with the goal of enabling researchers to investigate whether domain knowledge can similarly improve interpretability and steerability in genomics.

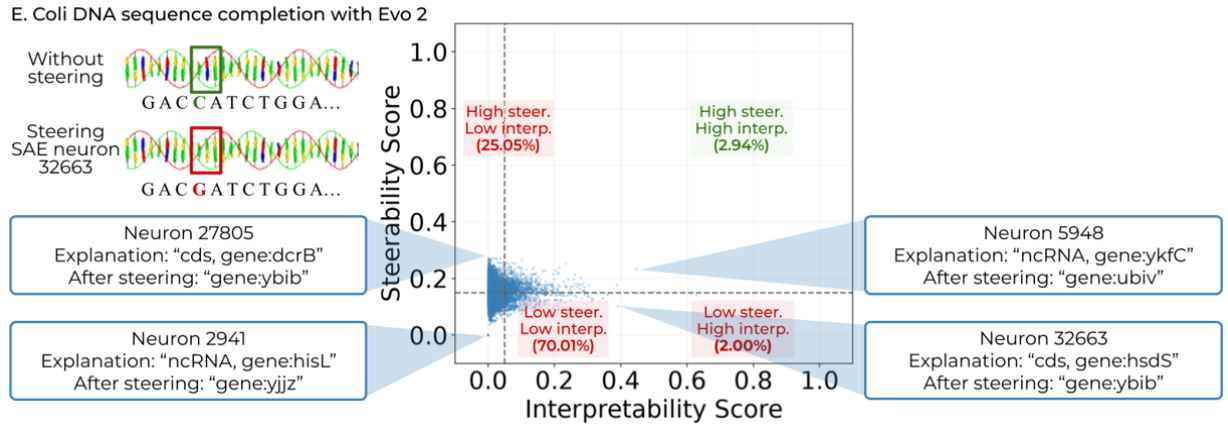

Figure 7: Interpretability versus steerability tradeoff in SAEs trained on Evo 2, a biological foundation model. The same qualitative trend observed in representative vision-language models also appears in this setting. Only a small fraction of SAE neurons exhibits both high interpretability and high steerability, revealing a tradeoff between these desirable properties. The research team is extending CB-SAEs to biological foundation models to address this limitation.

This work represents one step toward a broader vision for human-centered AI at LLNL. Beyond developing new interpretability algorithms, the team is building interactive tools for visualization [2], concept steering, and continual learning [3] that make interpretability actionable for domain scientists. Looking ahead, they aim to move beyond explaining isolated concepts within individual layers toward understanding the computational pathways that connect them across an entire model. They are also exploring how these ideas extend beyond open-weight foundation models to proprietary commercial models and increasingly agentic AI systems. As AI evolves from a prediction engine into a scientific collaborator, interpretability must evolve with it, providing scientists and foundation models with a shared language for discovery.

[1] A. Kulkarni, T.W. Weng, V. Narayanaswamy, S. Liu, W.A. Sakla, and K. Thopalli. "Interpretable and steerable concept bottleneck sparse autoencoders." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. doi.org/10.48550/arXiv.2512.10805.

[2] X. Yan, S. Liu, K. Thopalli, and B. Wang. "Visual exploration of feature relationships in sparse autoencoders with curated concepts." In *Conference on Neural Information Processing Systems, Workshop on Mechanistic Interpretability*, 2025. doi.org/10.48550/arXiv.2511.06048.

[3] S. Kundargi, K. Thopalli, and T. Gokhale. "SACK: sequentially acquiring concept knowledge to guide continual learning." In *Fifth Conference on Lifelong Learning Agents (CoLLAs)*, 2026. Read at OpenReview.

AI/ML & Applications | CompilerGPT: Leveraging Large Language Models for Compiler Optimization Reports

Contact: Peter Pirkelbauer and Chunhua Liao

Compilers translate source code into optimized machine code that can be executed by computers. The code generation and optimization passes are opaque to software engineers, but compilers such as Clang/LLVM and GCC (GNU compiler collection) can generate optimization reports to make the optimization process more transparent. However, compiler optimization reports are often hard to understand, requiring considerable expertise to interpret and act upon. This complexity hinders the effective utilization of compiler optimization capabilities, limiting their potential impact on software development.

CASC researchers Peter Pirkelbauer and Chunhua Liao developed CompilerGPT [1] to address this critical challenge in software optimization by bridging the semantic gap between cryptic compiler optimization reports and actionable code improvements. Compiler reports use highly technical language, requiring significant expertise to interpret them correctly. CompilerGPT leverages LLMs to analyze these reports and automatically rewrite code to address identified optimization opportunities. CompilerGPT implements a proto-agentic design with a configurable compile–optimize–evaluation loop that automates the interaction between compilers, LLMs, and user-defined evaluation harnesses.

CompilerGPT's workflow follows an iterative process as shown in Figure 8. The input code is compiled to ensure validity and tested to establish a baseline. Then the compiler generates an optimization report identifying missed opportunities and bottlenecks. CompilerGPT creates prompts by embedding the code, optimization report, and performance metrics. The full conversation history is maintained to enable successive refinements.

The LLM is tasked with analyzing the report, prioritizing high-impact issues, and optimizing the code. The generated code is compiled and tested using a user-provided evaluation harness. On failure, CompilerGPT prompts the LLM to fix errors by including error messages from the compiler or test harness. After reaching the iteration limit, CompilerGPT summarizes its findings.

CompilerGPT is a customizable framework where users can configure the compiler; the LLM model (supporting various providers including locally deployed models like llama and ollama, LLNL's internal systems, and external commercial providers like OpenAI, Anthropic, and OpenRouter); the evaluation harness (user-provided tests that verify correctness and measure runtime); and the prompts. The default prompt uses chain-of-thought style to break down complex optimization tasks, prioritize issues, choose the most high-impact issue, identify related code sections, and transform the code accordingly. To address the limited context window size in LLMs, users can select a specific code region for targeted optimizations. CompilerGPT then focuses on this selected code in its prompt and trims optimization reports to include only relevant, non-duplicate messages.

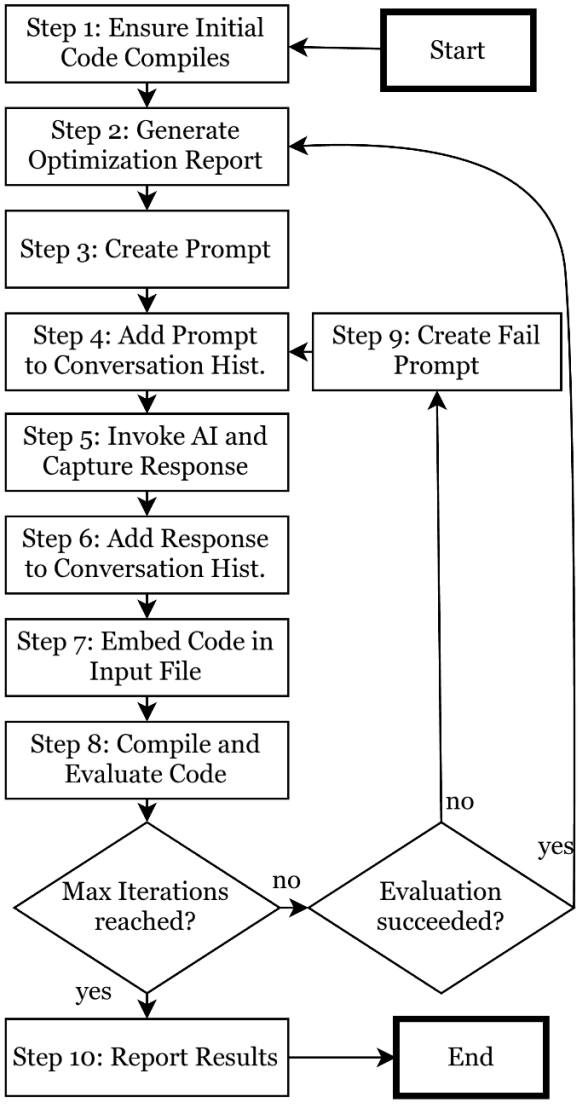

Figure 8: CompilerGPT workflow.

CompilerGPT was evaluated on five benchmarks, testing two compilers (Clang 18.1.8 and GCC 12.2.1) with two AI models (GPT-4o and Claude Sonnet 3.7, the leading models in Spring 2025). The benchmarks included Naive Matrix Multiplication (sequential, 25 lines), Parallel Prefix Sum (OpenMP-kernel, 25 lines), Smith-Waterman (OpenMP wavefront algorithm, 110 lines), and two NASA OpenMP benchmarks: NAS-FT (Fast Fourier Transform, 332 lines) and NAS-BT (Block Tri-diagonal solver, 422 lines). Experiments were conducted on an Intel Xeon Gold 6248R CPU at 3.00GHz with six iterations of the compile-optimize-evaluation loop. Each experiment was repeated five times.

As Figure 9 shows, the results demonstrated significant speedups. The best performance improvement of 6.5x was obtained on Parallel Prefix Sum with Sonnet and GCC. CompilerGPT optimized vector copying, hoisted allocations, switched to raw pointers, and reduced OpenMP overhead. Matrix Multiplication achieved 3.1x speedup through progressive blocking and loop unrolling, Smith-Waterman showed 1.4x improvement with Sonnet and Clang by identifying thread contention bottlenecks, and NAS Benchmarks

demonstrated modest 1.1x improvements. Runtime ranged from 4 to 23 minutes with costs of $0.01 to $1.31 per run.

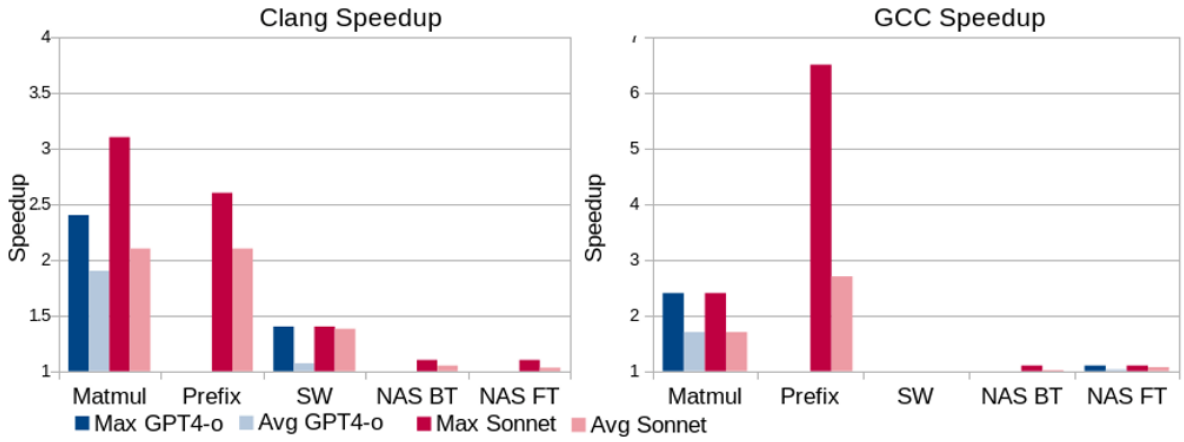

Figure 9: Results of CompilerGPT evaluation on five benchmarks, testing two compilers with two AI models.

The current version of CompilerGPT requires significant user involvement, such as code region identification, evaluation harness provision, and result interpretation. Future work focuses on designing CompilerGPT as an autonomous agent that would leverage external tools for profiling, automated test generation, and systematic optimization exploration to significantly reduce user interaction. CompilerGPT is available as open-source software.

[1] P. Pirkelbauer and C. Liao. "CompilerGPT: leveraging large language models for analyzing and acting on compiler optimization reports." In *Proceedings of ISC High Performance, Lecture Notes in Computer Science*, 2025. doi.org/10.1007/978-3-032-07612-0_25.

CASC Newsletter Sign-Up

Was this newsletter link passed along to you? Or did you happen to find it on social media? Sign up to be notified of future newsletters.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-MI-2023654. Edited by Ming Jiang.