Machine Learning for Computational Science and Engineering (2025) 1:23
https://doi.org/10.1007/s44379-025-00022-2

RESEARCH

Constitutive scientific generative agent (CSGA): Leveraging large language models for automated constitutive model discovery

Marius Tacke[1] · Matthias Busch[2] · Kartik Bali[1] · Kian Abdolazizi[2] · Kevin Linka[2] · Christian Cyron[1,2] · Roland Aydin[1,2]

Received: 28 March 2025 / Accepted: 28 April 2025 / Published online: 29 May 2025

Abstract
Data-driven approaches for constitutive modeling enable rapid, automated generation of models that predict a material's mechanical response under load. Integrating theoretical knowledge into these approaches, which are then called grey-box approaches, can improve sample efficiency, extrapolation capability, and interpretability, albeit typically at the cost of experts required to use them. Recently, general-purpose large language model (LLM)-based scientific discovery methods have emerged as user-friendly approaches to scientific discovery. In this work, we compare two representatives of these paradigms: highly specialized constitutive artificial neural networks (CANNs) and the general LLM-based scientific generative agent (SGA) to evaluate current LLM capabilities in constitutive modeling. In addition, we introduce the constitutive scientific generative agent (CSGA) to combine both approaches' strengths by enriching the SGA's prompts with domain-specific data and materials theory. We compare CANN, SGA, and CSGA on three benchmark problems by assessing their accuracy in predicting stress responses under prescribed strain conditions. While our results show that CANNs remain the most accurate approach overall, the CSGA significantly outperforms the SGA and demonstrates the promise of specialized LLM-based methods for constitutive modeling. Moreover, the CSGA's intuitive plain text interface and the full interpretability of the generated constitutive models make it a practical, accessible complement to existing approaches.

Keywords Constitutive modeling · Brain tissue deformation · Constitutive artificial neural network (CANN) · Large language model (LLM) · Scientific generative agent (SGA)

1 Introduction

A constitutive model is required to predict the mechanical behavior of a material under load by relating mechanical strain to mechanical stress. Constitutive modeling was initially based on empirical derivation of symbolic expressions and delivered models such as the Mooney-Rivlin model [1, 2], the Neo-Hookean model [3], or the Ogden model [4], describing the complex nonlinear mechanical behavior of materials such as rubbers, polymers, and biological tissues.

With increasing resources for data collection and computation, data-driven concepts have emerged as a response to the time-intensive and laborious nature of traditional methods. Early approaches, such as the distance-minimizing method [5, 6], utilize experimental data directly to solve mechanical problems, minimizing the distance between the current state and the data while satisfying conservation laws. Alternatively, researchers have experimented with leveraging artificial neural networks as black box models for constitutive modeling [7, 8]. Spline-based methods have become established as another black-box technique for constitutive modeling. Sussmann and Bathe [9] initiated the use of spline interpolations to model incompressible isotropic hyperelastic materials. Building on this, Latorre and Monatáns [10] extended the spline-based model to incompressible transversely isotropic materials, such as biological tissues and

✉ Marius Tacke
marius.tacke@hereon.de

✉ Christian Cyron
christian.cyron@hereon.de

✉ Roland Aydin
roland.aydin@hereon.de

[1] Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Strasse 1, 21502 Geesthacht, Germany

[2] Institute for Continuum and Material Mechanics, Hamburg University of Technology, Eissendorfer Strasse 42, 21073 Hamburg, Germany

engineered composites, and Dal et al. [11] leveraged spline interpolations for rubber-like materials. These approaches have in common that they automate the derivation of constitutive models from data, eliminating the need for manual model derivation. The strength of these purely data-driven approaches lies in their flexibility, as they can capture a wide range of experimental data. However, these methods require a substantial number of data points since all knowledge must be derived from the data itself. They are also highly sensitive to data quality and tend to extrapolate poorly beyond their training data due to the lack of additional information. Additionally, these models are typically not directly interpretable.

To tackle these challenges, researchers have developed so-called grey box models by integrating known physical laws, constraints, and general insights into black box machine learning models, which enhances their interpretability and adherence to these principles [12]. Physics-informed neural networks (PINNs) allow embedding the distance to the fulfillment of physical laws into the loss function [13]. Physics-augmented neural networks (PANNs) ensure by their construction that their predictions adhere to fundamental physics, enhancing the reliability of hyperelastic material models [14]. Mechanics-informed artificial neural networks (MIANNs) incorporate mechanical laws directly into the training process [15]. Constitutive artificial neural networks (CANNs) combine classical materials theory with machine learning algorithms [16–18]. Their specialization, viscoelastic constitutive artificial neural networks (vCANNs), integrates generalized Maxwell models with neural networks to capture nonlinear viscoelastic behavior [19]. The fusion-based constitutive model (FuCe) additively combines a classical constitutive model with a deep learning correction [20]. While the classical model provides a robust baseline for material behavior, the neural network component learns the discrepancies between this baseline and the specific material response, sacrificing some degree of interpretability for satisfying physical constraints. Neural ordinary differential equations (NODEs) use ordinary differential equations within neural networks to model dynamic systems [21]. The theoretical principles included in these approaches reduce the number of required data points and increase their extrapolation capabilities compared to full black-box models. However, each approach keeps an uninterpretable black-box model as part of the framework, which makes them lack full interpretability and therefore full trustworthiness, which is crucial in fields like biomedical engineering.

Addressing this need for explicit interpretability, recent works focus on automatically deriving constitutive models as symbolic expressions from data. A natural approach on this path is symbolic regression, which searches the space of mathematical expressions to identify models that best fit the data without assuming a predefined form [22, 23]. Flaschel et al. [24] introduced an approach that derives constitutive models in the form of parameterized mathematical expressions from displacement measurements of deformed bodies and global force data. The resulting material model is fully transparent and interpretable. After validating the framework in the complex task of modeling the behavior of path-dependent elastoplastic materials [25], the authors named the framework efficient unsupervised constitutive law identification and discovery (EUCLID). The framework was extended using Bayesian networks for model selection [26] and tested with neural networks as constitutive model cores, increasing the flexibility in adaptation to new materials [27]. Inspired by the Kolmogorov-Arnold representation theorem [28], Liu et al. [29, 30] introduced Kolmogorov-Arnold networks (KANs) as an alternative to multi-layer perceptrons (MLPs). Unlike MLPs, KANs use learnable activation functions on edges instead of fixed ones on nodes. Building on KANs, Abdolazizi et al. [31] developed constitutive Kolmogorov-Arnold networks (CKANs), which specialize in providing symbolic expressions without any remaining black-box elements, making them straightforward to interpret.

From the initial manual derivation of constitutive models, we have advanced to sophisticated techniques such as PANNs, CANNs, NODEs, symbolic regression, EUCLID, and CKANs. These models offer near-complete interpretability, high accuracy, and significant automation. However, they all share a common limitation: the need for extensive expert knowledge to prepare data in accordance with the model requirements, select the correct model configuration, and interpret the results. Large Language Models (LLMs) present a promising solution to this challenge. With their extensive pretraining corpus, LLMs have the potential to contribute to a wide range of research fields and can effectively be focused on desired specializations, as Han and Jiang showed [32]. Chen et al. [33] developed an LLM-based bilevel optimization approach for biophysical sequence optimization. Ma et al. [34] proposed a comparable, but more general framework for scientific equation discovery, which they call scientific generative agent (SGA). Both methods utilize a bilevel optimization: the outer loop generates sequence candidates or uncalibrated hypotheses using LLMs, while the inner loop iteratively refines these candidates through parameter optimization. Another comparable approach is called OpenFOAMGPT and was introduced by Pandey et al [35] for simulating fluid dynamics at multiple optimization layers. These approaches share that the complex candidate or model generation is covered by the LLM and only the parameterization is conducted by a second level or layer of optimization.

This work bridges the gap between highly automated, easy-to-use, and fully interpretable LLM-based approaches, which currently lack specialization and accuracy, and highly specialized expert methods such as CANNs, EUCLID, and other frameworks discussed above. While LLMs have shown progress in conversational and analytical tasks, a systematic

evaluation of their capabilities in constitutive modeling compared to specialized benchmarks like CANNs has been missing. Our initial objective was to fill this gap by systematically comparing the LLM-based SGA with the CANN on various constitutive modeling tasks.

During this comparison, we discovered the value of continuum mechanics theory, incorporated in CANNs, for the SGA to generate useful constitutive models. Thus, our contribution includes not only this comparison but also the introduction of the specialized constitutive scientific generative agent (CSGA), which combines the advantages of both worlds. In the CSGA, an LLM, provided with experimental data and materials theory, suggests suitable constitutive models, whose parameters are then optimized through a second level of optimization within the framework. This approach maintains the classical structure of fitting a preselected model to experimental data, but the task of model selection, previously designated to human experts, is now covered by an LLM. This LLM at the core of the CSGA lowers the barrier to creating constitutive models by offering an intuitive, text-based interface that requires minimal prior expertise. With the LLMs' growing capabilities in processing multimodal inputs, additional information does not even have to be text-based [36]. We demonstrate that CSGA outperforms SGA and is comparable to the near-perfect CANN performance, particularly in terms of extrapolation capabilities.

In the following Section 2, we provide a detailed account of the theoretical and practical background of our work. Section 3 introduces our CSGA concept in detail. The evaluation of the CSGA compared to SGA and CANN and the corresponding results are presented in Section 4, followed by a discussion in Section 5. Finally, we summarize the insights gained in Section 6.

2 Background

2.1 Continuum mechanics

In the framework of nonlinear continuum mechanics, as explained in [37], motion is described by a current position tensor $\mathbf{x}$ for each reference position tensor $\mathbf{X}$ in the body. We denote all tensors in bold. The deformation gradient tensor $\mathbf{F}$ is defined as the gradient of this motion with respect to the reference position:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \tag{1}$$

The deformation gradient's spectral decomposition provides the principal stretches λ_α with $\mathbf{n}_\alpha$ and $\mathbf{N}_\alpha$ as the corresponding principal directions in the current and reference configuration:

$$\mathbf{F} = \sum_{\alpha=1}^{3} \lambda_\alpha \mathbf{n}_\alpha \otimes \mathbf{N}_\alpha \tag{2}$$

As the deformation gradient captures not only deformation but also rotation, it is reasonable to introduce a pure deformation tensor such as the right Cauchy-Green deformation tensor $\mathbf{C}$:

$$\mathbf{C} = \mathbf{F}^T \mathbf{F} \tag{3}$$

In this work we only consider hyperelastic materials. These materials have a strain energy density function Ψ, from which the stress-strain relationship of a material can be derived. The strain energy density function Ψ defines the stored energy in a material as a function of the deformation. Since the strain energy density function Ψ is invariant to rotation, its dependency on the right Cauchy-Green deformation tensor $\mathbf{C}$ can be reduced to a dependency on the latter's invariants I_1, I_2, I_3:

$$I_1 = \mathrm{tr}(\mathbf{C}) \tag{4}$$

$$I_2 = \frac{1}{2}(\mathrm{tr}(\mathbf{C})^2 - \mathrm{tr}(\mathbf{C}^2)) \tag{5}$$

$$I_3 = \det(\mathbf{C}) \tag{6}$$

For incompressible materials, I_3 expresses the absence of any volume change as:

$$I_3 = 1 \tag{7}$$

Once the strain energy density function is known, the first Piola-Kirchhoff stress $\mathbf{P}$ can be derived as partial derivative of the strain energy density function Ψ with respect to the deformation gradient $\mathbf{F}$, modified by a pressure term $-p\mathbf{F}^{-T}$ to ensure perfect incompressibility. The hydrostatic pressure p needs to be determined with the boundary conditions.

$$\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}} - p\mathbf{F}^{-T} \tag{8}$$

2.2 Constitutive artificial neural network (CANN)

The CANN architecture precisely mirrors the theoretical background described in Subsection 2.1. As a constitutive model, the CANN's objective is to map strains to stresses, specifically, to map the deformation gradient $\mathbf{F}$ to the first Piola-Kirchhoff stress $\mathbf{P}$. On the input side, the deformation gradient $\mathbf{F}$ is reduced to its invariants or principal stretches. On the output side, the first Piola-Kirchhoff stress $\mathbf{P}$ is derived from the strain energy density function Ψ. This simplifies the mapping from an input tensor to an output tensor to

the mapping between two or three scalars to a single scalar. Additionally, deriving stress from strain energy allows for the enforcement of thermodynamic consistency.

The choice between using invariants or principal stretches as the characteristic of the deformation gradient to base the prediction of strain energy on significantly influences the CANN architecture. Linka et al. [16] used an invariant-based CANN to model the synthetic rubber dataset that will be used in this study as well. For isotropic materials, the general CANN architecture depicted in Figure 1 of [16] simplifies to a single feedforward neural network at its core. The architecture of this simplified model is schematically illustrated in Fig. 1. The first and second invariants of the right Cauchy-Green deformation tensor $\mathbf{C}$, I_1 and I_2, serve as inputs to a fully connected, feedforward neural network. This network comprises two hidden layers, each containing eight neurons. The third invariant, I_3, is neglected because it always equals one for perfectly incompressible materials. The activation function employed for each node is the softplus function:

$$\sigma(x) = \log[1 + \exp(x)] \tag{9}$$

The outputs from the second hidden layer are then weighted and summed to form the strain energy density function, Ψ.

St. Pierre et al. and McCulloch et al. [18, 38] found that principal stretch-based CANNs perform better on brain tissue data that will also be used in this study. We consider the best-performing CANN on this dataset as our benchmark, as identified in Pierre et al. [38]. In this model, the three principal stretches λ_1, λ_2, and λ_3 serve as inputs. These stretches are combined into n Ogden terms, with each term representing a sum of the three principal stretches raised to differently combined powers. The value of n indicates the complexity of the model. These Ogden terms are then weighted and summed to form the strain energy density function, Ψ. The best-performing model uses $n = 100$ Ogden terms and does not include L1 regularization in the loss function, which would penalize the size of the network weights and was also tested in this study.

2.3 Scientific generative agent (SGA)

In 2024, Ma et al. published a bilevel optimization framework called the scientific generative agent (SGA) [34]. Figure 2 provides an overview of the workflow: an initial prompt is sent to the LLM containing a problem description, some formal hints, and a code frame to be completed by the LLM. The LLM completes this code frame while leaving parameters to be optimized. This generation of an uncalibrated hypothesis constitutes the outer-level optimization of the SGA. The inner-level optimization involves hypothesis calibration, i.e., parameter optimization by an iterative

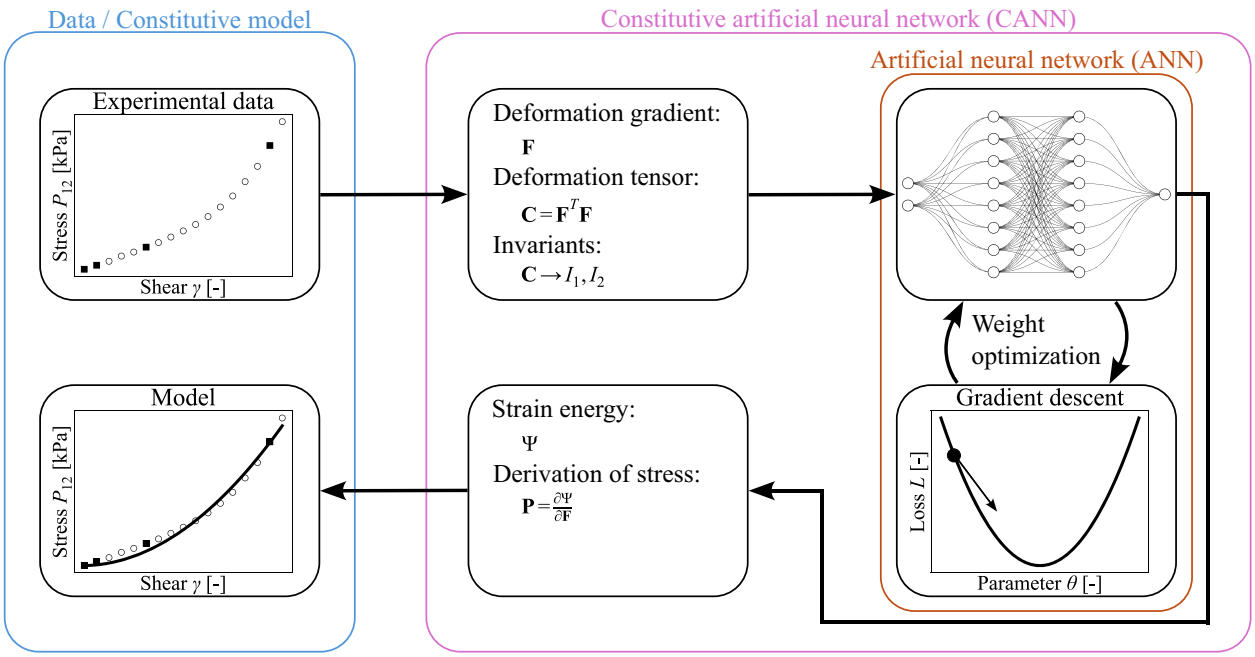

Fig. 1 Invariant-based CANN Architecture: the core of the constitutive artificial neural network (CANN) is an artificial neural network (ANN). Instead of directly using deformation and stress for fitting, deformations are formalized as deformation gradients, which are used to calculate the first two invariants I_1, I_2 of the Cauchy-Green deformation tensor $\mathbf{C}$. These invariants serve as inputs to the ANN. The output is the strain energy density function Ψ, from which the first Piola-Kirchhoff stress $\mathbf{P}$ can be derived. In this study, we focus for simplicity on a simple isotropic invariant-based CANN architecture. The general invariant-based CANN architecture is depicted in Figure 1 of [16]. The principal-stretch-based CANN architecture is comparable, with changes in the pre- and postprocessing, as described in Section 2.2

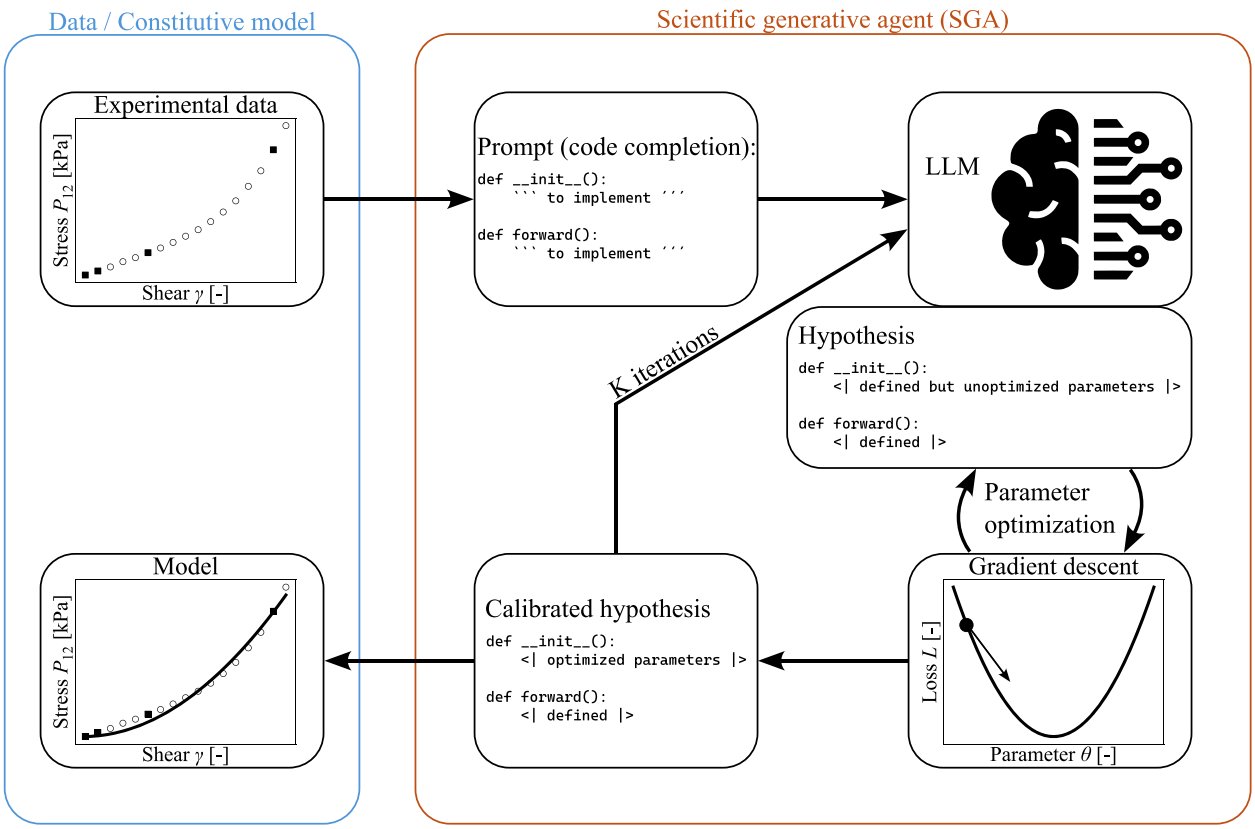

Fig. 2 Workflow of the scientific generative agent (SGA) approach: a large language model (LLM) generates a constitutive model suggestion within a code frame. This hypothesis is then calibrated using a standard optimization algorithm, such as stochastic gradient descent, and iteratively refined by resubmitting it to the LLM a predefined number of times

optimization algorithm, for which we use Adam. The calibrated hypothesis and its performance value are then appended to the initial prompt and sent for refinement through the combination of outer- and inner-level optimization until a predefined number of iterations is reached. The authors approached constitutive law discovery from motion observation, requiring the inner-level optimization algorithm to simulate the motion based on each suggested constitutive law and compare it to the observation. We focus on constitutive law discovery from tabular experimental data, allowing our version of the inner-level optimization to simply calculate the loss at the known data points for each suggested constitutive law without needing any simulation. Apart from that, we test the SGA closely following the original descriptions in [34] and only adapt the brief problem description to the new problem.

3 Method

When comparing CANN and SGA conceptually, we realize that the SGA lacks key information that is incorporated into the CANN architecture. The computations of the CANN are explicitly based on the deformation gradient's principal stretches or the right Cauchy-Green deformation tensor's invariants and use the strain energy density function to derive the first Piola-Kirchhoff stress tensor. Studies such as [18, 38] show how influential the choice between invariants and principal stretches can be. All this knowledge incorporated into the CANN architecture can be crucial for discovering effective constitutive models.

Considering the complementary nature of CANN and SGA, we wonder: can they be combined? Our approach of symbiosis is to integrate the knowledge incorporated in CANNs into the SGA by prompts and compare the resulting performance to CANNs and the original SGA. In one configuration, we instruct the SGA to first calculate the right Cauchy-Green deformation tensor's invariants and use them as the basis for the calculation of the first Piola-Kirchhoff stress tensor. In another configuration, we instruct the SGA to rely on the deformation gradient's principal stretches for the same calculation steps. In any case, we enrich the initial prompt with training data for a more informed suggestion of constitutive models, as we work with tabular data that can simply be added to a text prompt. Furthermore, after

observing that the suggested constitutive laws often violated the essential condition of no stress at no deformation, we explicitly add this condition to the prompt. We call the specialized version of the SGA, incorporating the combination of all these instructions, the constitutive scientific generative agent (CSGA). We illustrate the described extension of the SGA to the CSGA in Fig. 3.

We noticed that the LLMs sometimes appear less creative once the loss value they receive as feedback falls below 1, regardless of the metric. To address this, we multiply the loss provided to the LLMs as feedback for previous iterations to make it appear larger and maintain the LLMs' creativity.

For the comparison of all SGA versions, we use OpenAI's o1-preview as the LLM backbone. A key advantage of the SGA is that it automatically benefits from the groundbreaking developments of LLMs by simply switching to new LLMs. Therefore, we additionally compare OpenAI's GPT-4o, OpenAI's o3-mini (adjusted for high reasoning effort), Llama 3.1 405B, and DeepSeek-R1 as backbones for CSGA, our improved version of SGA. To account for the stochastic effects in LLM responses, we test each configuration 25 times on all three introduced datasets and present the best models in Section 4. A report on the variability across the 25 runs in each configuration can be found in Appendix A.

As we run all of these LLMs on external servers, we cannot measure their runtime in detail. However, we can provide an estimation of the computational cost of the tested approaches. On our standard desktop computer (Intel Core i9-11950H @ 2.60GHz, 64GB RAM, NVIDIA RTX A5000 with 24GB VRAM), training the CANN for 4000 epochs on the synthetic dataset, which includes 45 data points in total and 36 data points for training, takes around two minutes. The forward pass for a single data point takes approximately 6 milliseconds. Since the SGA and CSGA only differ in the prompts to the LLM, their computational costs are similar. Creating an SGA or CSGA model involves two stages: defining the model using an LLM and calibrating it with a standard optimization algorithm. We can measure the computational cost of closed-source LLMs such as o1-preview and o3-mini in terms of input and output tokens and their corresponding processing costs. Generating an SGA or CSGA model requires multiple calls to the LLM, as initially generated models are sent back for refinement. Including three previously generated models, the prompt is around 4,000 tokens long. We call

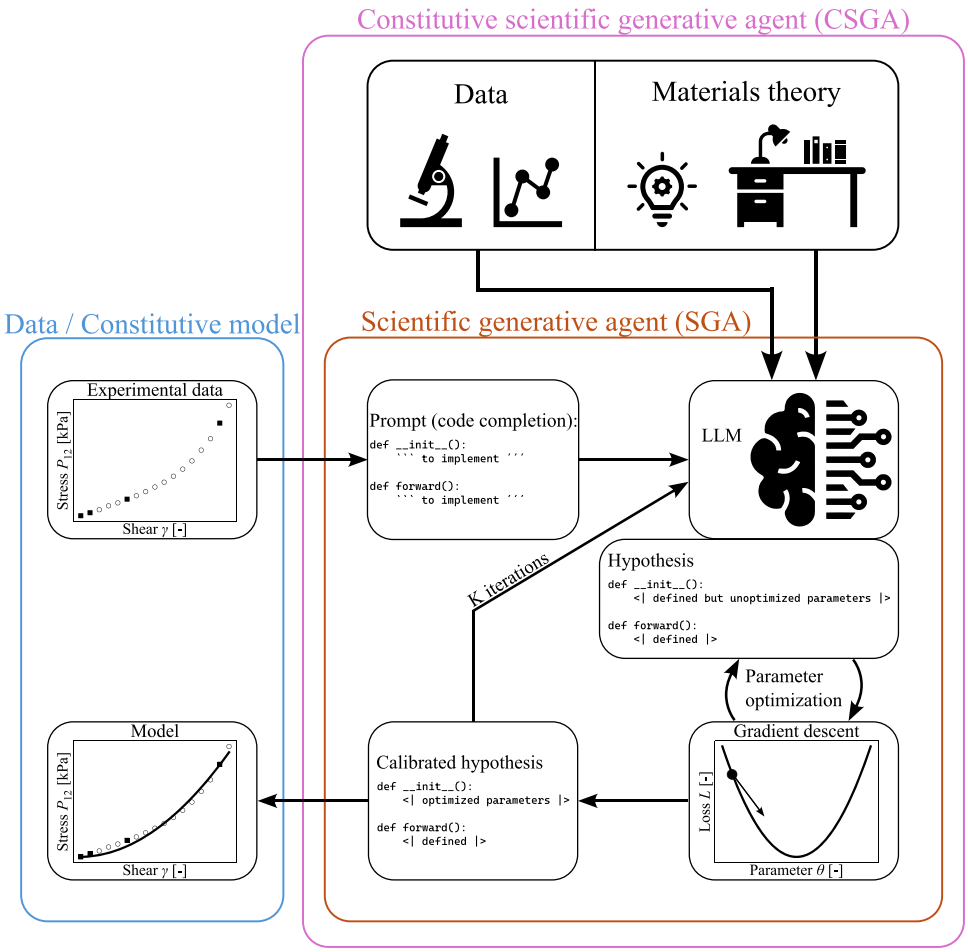

Fig. 3 Extension of the scientific generative agent (SGA) to the constitutive scientific generative agent (CSGA) approach: in this extended approach, we provide the large language model (LLM) with training data and expertise from continuum mechanics, including instructions to utilize the deformation gradient's principal stretches or the right Cauchy-Green deformation tensor's invariants. We refer to this enhanced method as CSGA. Additionally, we test intermediate steps between SGA and CSGA, such as providing only additional training data or solely instructing the LLM to use principal stretches or invariants

the LLM five times to generate one model, with each response typically around 2,000 tokens long. This totals 20,000 input tokens and 10,000 output tokens per model, translating to a cost of 90 USD cents for o1-preview and 7 USD cents for o3-mini. Fitting such a model for 200 epochs on the 36 training data points of the synthetic dataset takes around 2 seconds on our standard desktop computer. A forward pass takes approximately 0.04 milliseconds on the same computer. In summary, the CANN is computationally cheaper at model generation. However, this does not account for the cost of human experts in continuum mechanics required to set up the CANN. The SGA and CSGA approaches are cheaper at inference, which is likely more important as the models are ideally created once but used frequently.

The exact hyperparameter settings of SGA, CSGA and CANN are reported in Appendix B.

4 Results

4.1 Mechanical behavior of human brain tissue

The first problem we test all approaches on is modeling the constitutive behavior of brain tissue. Solving this problem is crucial for understanding the brain's mechanical properties, which can aid in developing better medical treatments and protective measures to heal or avoid brain injury. The problem is complex and has thus become an established benchmark problem with numerous approaches to create an accurate model [17, 18, 38–41]. Budday et al. collected the corresponding dataset by conducting experiments on ten human brains, seven male and three female, aged 54 to 81 years, within 60 hours post-mortem [39]. They cut the brains into slices and tested different regions. We use the work of St. Pierre et al. [38] as our model benchmark and focus on the gray matter tissue from the brain cortex. The tissue was tested under tension and compression, where each specimen was stretched in one direction, and under simple shear, where the specimen was sheared in one direction. Stretch is defined as:

$$\lambda = \frac{l}{l_0} \tag{10}$$

where l is the sample length in the current configuration and l_0 is the sample length in the reference configuration. Simple shear can be imagined as layers of the material sliding past each other. Shear is defined as the ratio of the displacement u between layers to the distance h between them:

$$\gamma = \frac{u}{h} \tag{11}$$

For each loading scenario, 17 data points are available. St. Pierre et al. [38] used all data points for training their model. For training the LLM-based approaches on the brain data, we separate three randomly selected data points as test data points to evaluate the ability of these models to interpolate between the training data.

Figure 4 shows the predictions of CANN, SGA, and CSGA for this dataset. We show the results from St. Pierre et al. [38] who used principal-stretch based CANNs with 100 Ogden terms, representing a highly complex form of the CANN. McCulloch et al., from the same research group, published a subsequent study on the performance of CANNs on the brain dataset, using only eight terms to maintain low complexity [18]. They achieved significantly less accurate models

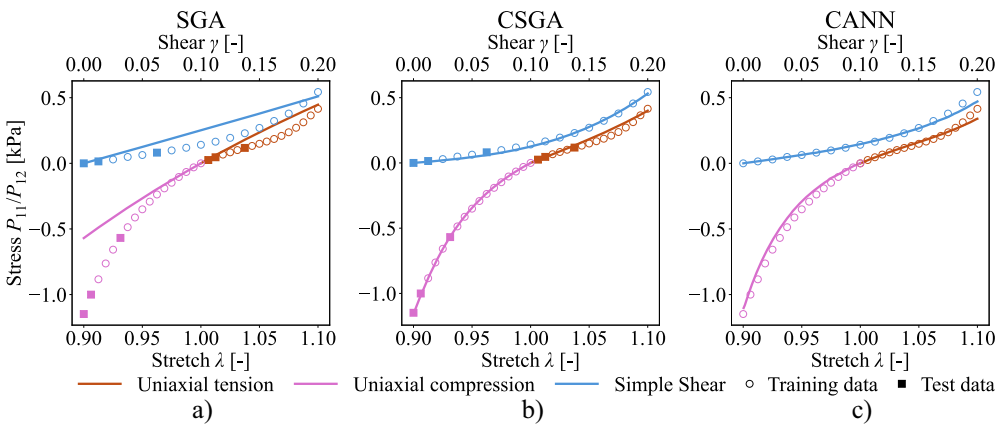

Fig. 4 Comparison of scientific generative agent (SGA) (a), constitutive scientific generative agent (CSGA) (b), and constitutive artificial neural network (CANN) (c) on the experimental brain tissue deformation dataset collected by [39] with the first Piola-Kirchhoff stress **P** at different levels of stretch and shear. P_{11} is the stress in stretch direction for tension and compression cases, while P_{12} is the shear stress for simple shear cases. The SGA struggles to accurately capture the experimental data, whereas CSGA and CANN perform significantly better

with R^2 scores of 0.84 for tension, 0.77 for compression, and 0.94 for simple shear loading. We do not impose any restrictions on complexity and compare SGA and CSGA with the best-performing, but highly complex, CANN version published by St. Pierre et al. [38]. This CANN version captures the experimental data perfectly, with R^2 scores ranging from 0.96 to 1.00. The SGA struggles significantly to capture the data, predicting only straight lines without coming close to capturing the correct curvature. Its R^2 values are only around 0.55 for the tension and compression loading scenarios and 0.73 for the simple shear loading scenario. The CSGA manages to capture all experimentally tested modes as well as the CANN. It only shows small errors in the tension loading scenario, where it reaches an R^2 score of 0.93. For the other two loading scenarios, it nearly perfectly matches the experimental data. Its MSE values are an order of magnitude smaller than those of the SGA approach. The reason for this surprisingly strong performance is likely that the suggested CSGA model decouples normal and shear loading by computing the singular value decomposition of the deformation gradient tensor. This raises the question of how well this model generalizes to multi-axial loading. Unfortunately, we cannot clarify this for either the CANN, the SGA or the CSGA due to the lack of experimental data on multiaxial loading of brain tissue. We will address this limitation with a synthetic dataset in Section 4.3.

4.2 Experimental data on rubber elasticity

We test SGA, CSGA, and CANN also on modeling the elastic behavior of rubber, using experimental data collected by Treloar [42]. Since its publication, this problem has become a benchmark for constitutive modeling [43–45] and was used to evaluate CANNs during their initial introduction [16]. Treloar's experiments involved stretching rubber specimens in various ways: uniaxial tension (stretching in one direction), equibiaxial tension (stretching in two directions simultaneously), and pure shear loading (stretching in one direction while keeping another direction steady). Each loading scenario provides 15 data points, and we maintain the same train-test split as the first work on this dataset with CANNs [16].

Figure 5 illustrates the predictions of SGA, CSGA, and CANN on this dataset. The SGA approach yields more accurate models compared to the brain dataset, yet significant inaccuracies persist. The CSGA outperforms the SGA substantially, delivering highly accurate models with R^2 values of 0.99, 0.97, and 0.98, and MSE values less than half those of the SGA. The explicit instruction to predict no stress at no strain, one difference between CSGA and SGA, proves important for the LLM-based approach, as only CSGA adheres to this constraint. The CANN results, as shown in [16], further improve upon the CSGA, providing fully accurate models with R^2 scores of 1.0 for each loading scenario.

4.3 Synthetic data on rubber elasticity

The third problem we use is modeling the constitutive behavior of a fictitious, isotropic, incompressible, rubber-like material, which was published with the introduction of the CANN [16]. Experimental data, such as those described above, represent the full complexity of reality, making them valuable benchmark problems. However, in experiments, it is often only possible to measure load and deformation along

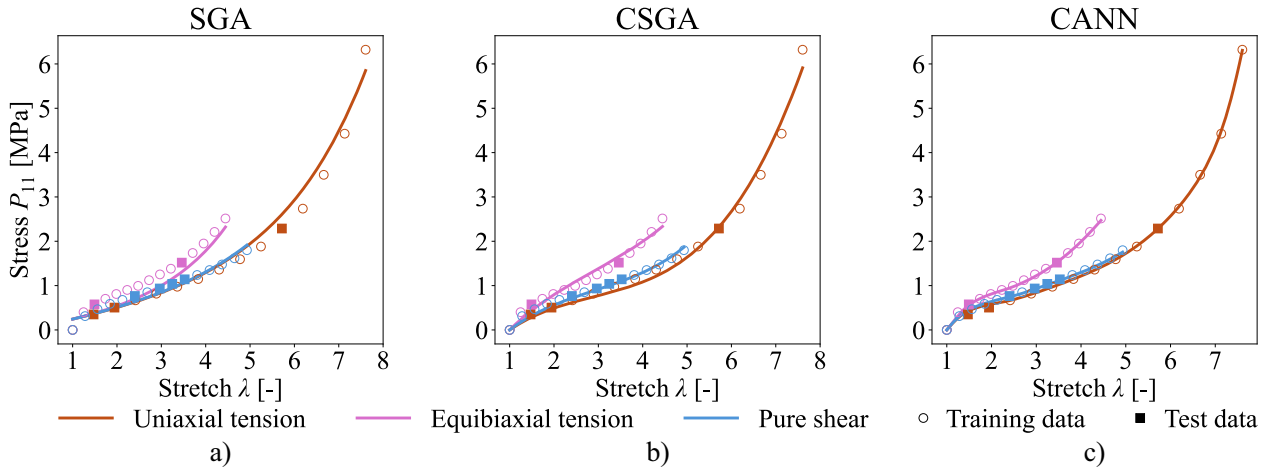

Fig. 5 Comparison of scientific generative agent (SGA) (a), constitutive scientific generative agent (CSGA) (b), and constitutive artificial neural network (CANN) (c) on experimental rubber dataset: this dataset, published in Treloar [42], describes the first Piola-Kirchhoff stress **P** at different levels of stretch λ for vulcanized rubber in uniaxial tension, equibiaxial tension, and pure shear loading scenarios. P_{11} is the stress in loading direction. The CSGA almost matches the perfect predictions of the CANN, while the SGA shows significant inaccuracies

a single axis. While modeling these simple loading scenarios can be challenging, the true difficulty lies in modeling multi-axial loading. The capability of a model to capture such multi-axial loading scenarios often cannot be evaluated with experimental data, as the ground truth in these scenarios remains unknown. This is why we also compare SGA and CANN in capturing the constitutive behavior of a fictitious material: here, we can train both models on uni-axial loading scenarios and evaluate their generalization to multi-axial loading scenarios. 15 samples are generated following the same loading scenarios as described in Section 4.2. We use the exact same train-test split as the original work introducing this dataset [16]. For the generation of the synthetic data, the Mooney-Rivlin strain energy function

$$\Psi = \sum_{i=1}^{3} c_{i0}(\mathbf{I}_\mathbf{C} - 3)^i + c_{0i}(\mathbf{II}_\mathbf{C} - 3)^i \tag{12}$$

with parameters $c_{10} = 1.6 \cdot 10^{-1}$, $c_{20} = -1.4 \cdot 10^{-3}$, $c_{30} = 3.9 \cdot 10^{-5}$, $c_{01} = 1.5 \cdot 10^{-2}$, $c_{02} = -2.0 \cdot 10^{-6}$, $c_{03} = 1.0 \cdot 10^{-10}$ is used. This strain energy function, defined for a fictitious material, maps the invariants of the right Cauchy-Green deformation tensor $\mathbf{C}$, which are derived from a given strain, into a corresponding strain energy Ψ. From this, the associated stress can be computed. This formulation allows for the generation of pairs of first Piola-Kirchhoff stress tensors $\mathbf{P}$ corresponding to arbitrary deformation gradients $\mathbf{F}$. The selected material parameters are chosen to resemble experimental observations [16].

Careful readers might notice that the data points plotted here differ slightly from the original data points [16]. After reproducing the synthetic dataset, we noticed a small error in the consideration of the hydrostatic pressure in the calculation of the first Piola-Kirchhoff stress tensor from the strain energy in the code corresponding to the first paper introducing CANNs [16]. We repeated the tests of the CANN with the corrected dataset and present the updated results in Table 1 and Fig. 6. They differ only slightly from the original results and do not change the original paper's message.

The predictions of SGA, CSGA, and CANN on this dataset are shown in Fig. 6. The CANN captures all three loading scenarios perfectly, achieving R^2 values of 1 for each scenario. In contrast, the SGA performs significantly worse. It only approximates the correct curvature for equi-biaxial tension. For the other two loading scenarios, it produces only straight lines and predicts a significant stress value at the point of no deformation, which is incorrect. The CSGA, instructed to ensure no stress occurs at no deformation, adheres to this condition. Furthermore, it captures equi-biaxial tension almost as well as the CANN. While it still predicts straight lines for uniaxial tension and pure shear, these lines are at a more suitable position. The superior performance of CSGA compared to SGA is also evident in the error values: The CSGA achieves higher R^2 correlation scores for each loading scenario, and its MSE values are again approximately half the size of the SGA's error values.

Our curiosity about the CSGA models' ability to generalize to multiaxial loading motivated us to compare all approaches on this synthetic dataset, where we can calculate the stress in scenarios of multiaxial loading, which can be illustrated by the so-called invariant plane I_1-I_2, first suggested by Treloar [46]. Figure 7 shows this invariant plane with the errors of CANN, SGA, and CSGA for the synthetic rubber dataset. The x and y coordinates are the first and second invariants of the right Cauchy-Green deformation tensor $\mathbf{C}$, defining the loading scenario. The color indicates the deviation from the correct stress value, specifically the first entry of the first Piola-Kirchhoff stress tensor $\mathbf{P}$. The plane is bounded by the uniaxial tension loading on the lower end and by the equi-biaxial tension loading on the upper end. The pure shear loading marks the angle bisector between them (note the unequal scaling of the axes). We cap the error values at 5% to ensure nuances in generalizability are visible. Larger errors, which mostly occur in the pure shear loading scenario, can be studied in Fig. 6. The CANN captures

Table 1 Comparison of R^2 and MSE values for constitutive artificial neural network (CANN), scientific generative agent (SGA), and constitutive scientific generative agent (CSGA) fitted on the brain tissue deformation, the experimental rubber, and the synthetic rubber dataset

	R^2			MSE		
	SGA	CSGA	CANN	SGA	CSGA	CANN
Brain. Uniaxial tension	0.56	0.93	0.96	5E-3	8E-4	6E-4
Brain. Uniaxial compression	0.53	1.00	0.99	6E-2	4E-5	2E-3
Brain. Simple Shear	0.73	0.99	1.00	7E-3	2E-4	5E-4
Exp. rubber. Uniaxial tension	0.98	0.99	1.00	6E-2	3E-2	1E-3
Exp. rubber. Equibiaxial tension	0.88	0.97	1.00	5E-2	1E-2	1E-3
Exp. rubber. Pure shear	0.95	0.98	1.00	1E-2	4E-3	4E-4
Synth. rubber. Uniaxial tension	0.80	0.87	1.00	2E-1	1E-1	3E-5
Synth. rubber. Equibiaxial tension	0.94	0.98	1.00	5E-2	2E-2	7E-6
Synth. rubber. Pure shear	0.86	0.93	1.00	4E-2	2E-2	5E-6

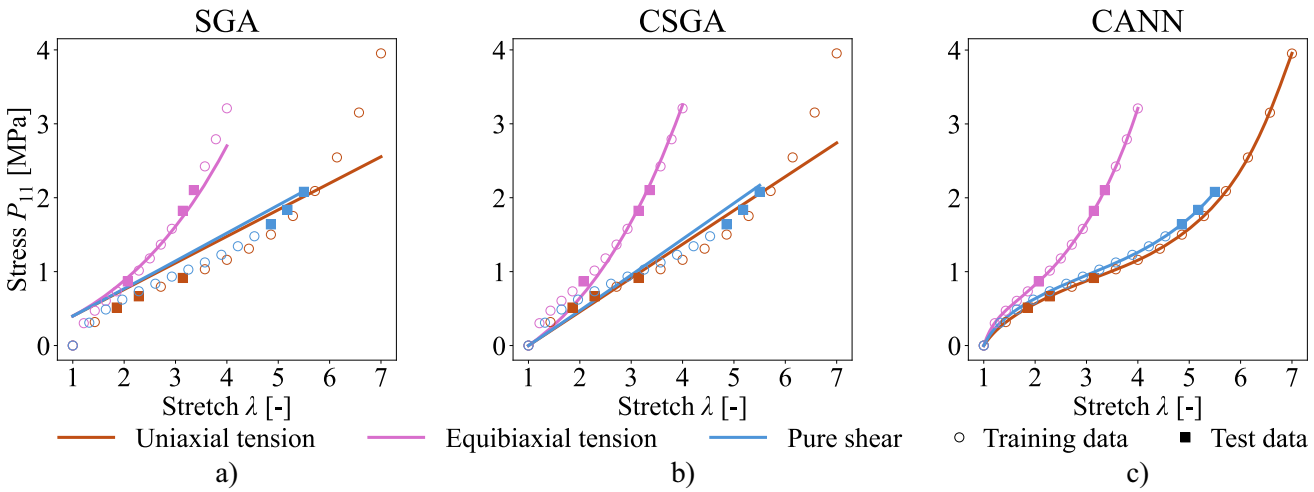

Fig. 6 Comparison of scientific generative agent (SGA) (a), constitutive scientific generative agent (CSGA) (b), and constitutive artificial neural network (CANN) (c) on synthetic rubber data: this dataset, introduced in Linka et al. [16], describes the first Piola-Kirchhoff stress **P** at different levels of stretch λ for a fictitious, rubber-like material in uniaxial tension, equi-biaxial tension, and pure shear loading scenarios. P_{11} is the stress in loading direction. The CANN captures the generated data perfectly, while the CSGA shows considerable shortcomings but still significantly outperforms the SGA

multiaxial loading perfectly and only shows weaknesses for larger extrapolation. SGA and CSGA generalize surprisingly well: for the majority of multiaxial loading scenarios, their predictions deviate from the true stress values not significantly further than the CANN's predictions. The SGA shows large errors for the pure shear loading scenario and for small deformations, as it incorrectly predicts a significant stress value at no deformation. The CSGA avoids these initial errors completely and only shows larger errors for the pure shear loading scenario. Remarkably, both SGA and CSGA extrapolate better beyond the training data than the CANN.

4.4 Different stages from SGA to CSGA and different LLMs

To understand the impact of each single change from SGA to CSGA, we also test multiple intermediate configurations on the experimental brain and the synthetic rubber dataset. We incrementally add constitutive modeling knowledge from the CANN to the SGA and observe the improvements. The correlation scores achieved by each configuration on both datasets are shown in Fig. 8b). We find that using the deformation gradient's invariants works better for the synthetic

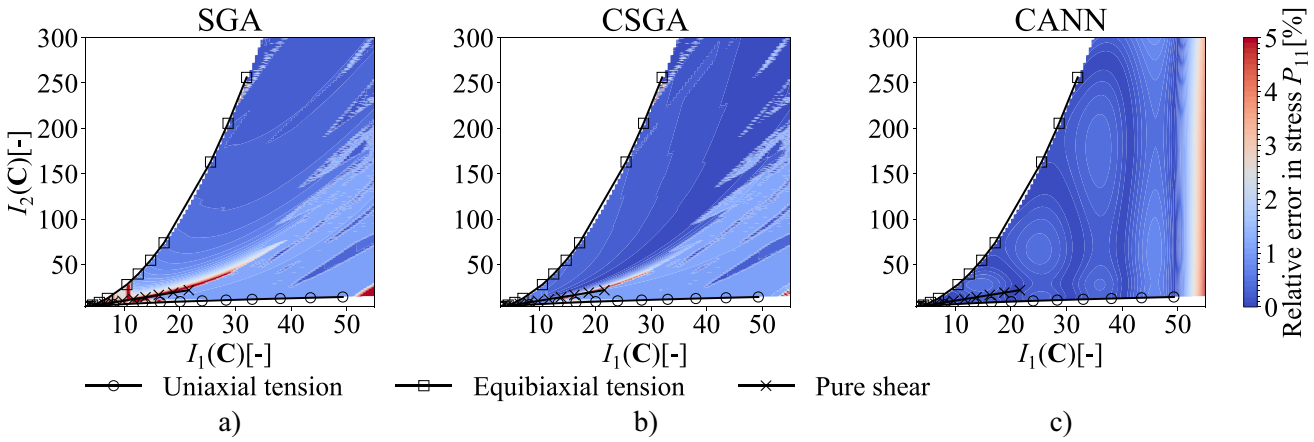

Fig. 7 Comparison of scientific generative agent (SGA) (a), constitutive scientific generative agent (CSGA) (b), and constitutive artificial neural network (CANN) (c) on invariant plane for synthetic rubber data introduced in Linka et al. [16] with the first Piola-Kirchhoff stress **P** at different levels of stretch for a fictitious, rubber-like material. The plane spanned by the first and second invariant I_1, I_2 of the right Cauchy-Green deformation tensor **C** covers any biaxial loading of a material. As this dataset describes a fictitious material, we can calculate the true first Piola-Kirchhoff stress value P_{11} for any loading situation and compare it to the models' predictions. The CANN generalizes best to multi-axial loading, but SGA and CSGA extrapolate better beyond the training data

rubber dataset, while using the deformation gradient's principal stretches works better for the brain dataset. These findings align with the observations from the CANN hyperparameter search by McCulloch et al. [18].

We also test different LLMs as the outer-level optimization unit of the CSGA on the experimental brain and the synthetic rubber dataset. Figure 8a) shows the correlation scores achieved by the CSGA on the two datasets using each of these LLMs. The center of each radar chart represents a correlation score of 0, while the outer edge represents an optimal correlation score of 1. The larger the polygon, the more accurate the corresponding prediction. For the synthetic rubber dataset, o1-preview delivers the best performance, while for the brain dataset, o3-mini performs the strongest. We observe the same result when testing the SGA with different LLMs as backbones. Comparing the LLMs, we observe a clear trend that models focused on reasoning, such as o3-mini, o1-preview, and DeepSeek-R1, outperform the other models. GPT-4o and Llama 3.1 405B clearly lag behind.

5 Discussion

First, we contrast the differences in methodology between the SGA and the CSGA with their respective performances. The CSGA is the specialization of the LLM-based SGA framework for general scientific discovery. The primary technical difference between the CSGA and the SGA lies in the incorporation of training data and materials theory hints added to the prompt. Based on our evaluation of the predictions of the SGA, CSGA, and CANN on the experimental brain tissue data and the experimental and synthetic rubber data, our introduction of the CSGA bridges the gap between general LLM-based methods like the SGA and specialized constitutive modeling methods like CANNs, not only in terms of methodology but also in terms of performance. The CSGA consistently outperforms the SGA across all datasets and nearly matches the performance of the CANN on the two experimental datasets. To understand the CSGA's superior performance, we analyzed intermediate steps between the SGA and the CSGA, as shown in Fig. 8. While adding training data to the prompt is beneficial, informing the constitutive model framework with a specific functional basis is significantly more influential. In practical terms, guiding the LLM to build its constitutive law on invariants gives it a fully coordinate-free, symmetry-respecting toolkit, as functions of the invariants automatically satisfy objectivity and isotropy and can be mixed and matched to capture very complex stiffening behaviors. That extra expressivity, however, comes at the cost of a non-convex optimization problem in which multiple local minima and strong collinearity between terms can make it hard to find a unique global fit. By contrast, steering the model toward principal stretches effectively fixes a family of Ogden-type exponents in advance, turning training into a convex regression with a single robust solution. However, this solution can only capture behaviors that lie within that

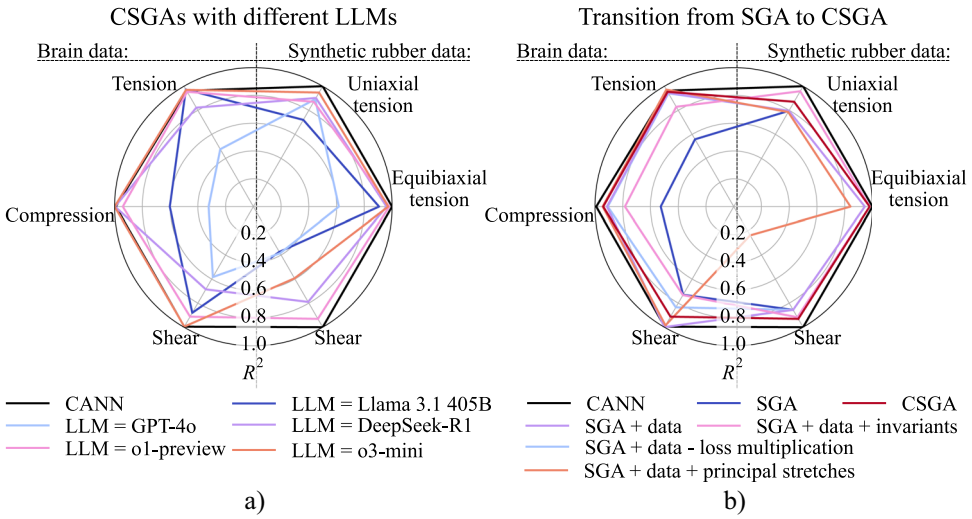

Fig. 8 Comparison of different large language models (LLMs) as backbones of the CSGA (a) and different intermediate steps between scientific generative agent (SGA) and constitutive scientific generative agent (CSGA) (b) by R^2 values: the larger the polygon, the closer the corresponding R^2 values to 1, indicating better performance. The constitutive artificial neural network (CANN) delivers near-perfect performance for all loading scenarios of the experimental brain and synthetic rubber dataset. In comparing LLMs, we observe that models focused on reasoning (DeepSeek-R1, o1-preview, o3-mini) significantly outperform the standard models GPT-4o and Llama 3.1 405B. The CSGA approach differs from the SGA approach by providing the LLM not only with the task but also with exemplary data and materials theory. The performance of the invariant-based approach versus the principal-stretch-based approach depends heavily on the dataset. Additionally, more information leads to better models, with CSGA significantly outperforming SGA

narrower exponent set. Because different materials (and even different loading modes on the same tissue) may emphasize tension compression asymmetry, shear-dominated stiffening, or higher-order invariant effects, it is very difficult to know a priori which representation will yield the best trade-off between bias and variance. A direct remedy, of course, is simply to have the CSGA try an invariant-based form and a principal-stretch form one after the other and pick the better fit, which is exactly what we did in our experiments. The SGA, despite having essentially enough compute, iterations, and attempts, never performed this straightforward model-class sweep, which only underscores how critical even minimal domain guidance can be for efficient, accurate constitutive discovery.

After analyzing how expert knowledge in materials theory is key for the CSGA to outperform the SGA, we still claim that the CSGA is more intuitive to use compared to specialized methods, which may seem contradictory at first glance. First, there is a difference between guessing the optimal path within continuum mechanics to a constitutive model, which directly relates to entering this as text in the CSGA prompt, and setting up a fully functional specialized architecture such as a CANN based on this guess. Second, if one lacks this initial guess, an LLM can generate different paths to try, trading off computational cost for expert knowledge. Third, with the groundbreaking improvements in LLMs and CSGA's ability to switch to new LLMs without further changes, we consider CSGA only a snapshot within this dynamic development and expect the importance of providing expert knowledge to reduce over time.

When comparing the CSGA with the CANN, it is important to remember that the CANN is a highly specialized architecture, integrating decades of research on constitutive modeling into a neural network. For the brain dataset, we selected the best-performing model from multiple studies conducting extensive hyperparameter searches [17, 18, 38]. The CSGA has two key advantages and one key disadvantage compared to the CANN. First, configuring, setting up, and training the CANN on custom data requires a deep understanding of its architecture and the dataset. In contrast, the CSGA offers high flexibility via an intuitive interface in plain text, making it straightforward to use. If the model does not perform as desired, adjustments can easily be made to the prompt. For instance, when the CSGA incorrectly predicted significant stress at no deformation, we added a corresponding hint to the prompt, solving the issue - something certainly more challenging to implement in a CANN. Second, the CSGA models are fully interpretable. They are parameterized closed-form equations implemented in Python code, automatically described with comments, making them easy to understand, trust, and refine. In Appendix C, we exemplarily present the code for the best models for the experimental brain and the synthetic rubber dataset, with each calculation step classified within the framework of continuum mechanics. CANNs partially consist of black-box neural networks, constraining their interpretability. However, recent developments like CKANs [31] have largely addressed the lack of interpretability. The key disadvantage of the CSGA is the superior accuracy of CANN models. Across all tested datasets, CANN models consistently demonstrate significantly higher accuracy. Consequently, those familiar with specialized techniques are likely to achieve superior results. However, in practice, when no expert is available, generating a constitutive model with LLM-based approaches like the CSGA may be the only viable option.

Referring to Section 1, we comment on the position of the CSGA within the related frameworks EUCLID [24] and CKANs [31], although we have not conducted a quantitative comparison of these approaches with the CSGA in this study. All three approaches generate explicitly interpretable constitutive models from data. CKANs exhibit robust generalization capabilities because they are built as a regularized composition of univariate functions that, at sufficient size, can accurately capture any continuous multivariate function, which is asserted by the Kolmogorov-Arnold representation theorem [47]. A second strength is that the interpretability of each constitutive model is ensured by a symbolic post-processing step. This complete interpretability is likewise shared with the EUCLID approach. EUCLID's additional strength is the vertical extension of the data-driven modeling process. Conventional approaches, including the CSGA, typically require inferring stress from global load measurements using assumptions about cross-sectional properties, which is limited to simple tests like uniaxial tension and often prone to inaccuracies. EUCLID bypasses this step of stress inference by directly employing strain data and global load measurements, which enhances practical applicability. However, this vertical framework complicates direct comparisons with related methods and increases its overall complexity, potentially posing challenges for non-expert users. As with CKANs, expert handling remains important for the effective application of this approach. The CSGA utilizes LLMs to handle the complexity of deriving constitutive models from data, making it more straightforward and user-friendly. The adaptability of the CSGA is a significant advantage, as the framework can easily integrate more powerful LLMs as they become available. Additionally, the CSGA's ability to incorporate additional information, including non-quantifiable data and multimodal inputs, is unique in constitutive modeling. In terms of performance, while a direct head-to-head comparison has not been performed in this study, existing research [24, 25, 31] suggests that EUCLID and CKANs will likely outperform the CSGA in terms of accuracy. EUCLID is a well-established framework with multiple studies supporting its efficacy, whereas CSGA represents the first specialization of the general SGA

approach, indicating significant potential for further specialization and improvement.

Finally, we would like to discuss some LLM-specific aspects: LLMs typically involve prompt engineering. In this study, we use prompts from the original SGA study [34] without refinement. We tried to increase LLM creativity by multiplying the loss provided as feedback on previous suggestions, but this turned out to be ineffective, as shown in Fig. 8. To understand the impact of different constitutive modeling paths, such as using invariants or principal stretches, we instructed the LLM to follow these paths and evaluated the results. We believe this approach does not constitute prompt engineering overhead but rather simulates the scenario of modeling a material without prior knowledge of effective methods. By experimenting with different paths, we can compensate for the lack of theoretical knowledge through computational resources.

Among the LLMs we tested, OpenAI's closed-source models, o1-preview and o3-mini, perform best, though they introduce usage barriers and limit reproducibility to some extent. However, the open-source LLM DeepSeek R1 nearly matches their performance. Our work represents a snapshot of a rapidly evolving field, with new and more powerful LLMs being released weekly. Currently, OpenAI's LLMs appear to be slightly ahead, but slightly less accurate results are possible today with models like DeepSeek R1, and we anticipate that similar or even superior results will be achievable with open-source LLMs in the near future.

6 Conclusions

In this study, we benchmarked the general LLM-based SGA against the specialized CANN. As anticipated, the CANN achieved near-perfect accuracy across our benchmarks, while the general SGA underperformed. By augmenting SGA prompts with domain-specific data and materials theory to create the specialized CSGA, we substantially closed the accuracy gap: The CSGA almost matches the CANN on experimental datasets and demonstrates strong generalization on the multiaxial evaluation with synthetic data. Although the CANN remains more accurate and is highly automated, its complex setup and architecture pose challenges for users. In contrast, CSGA's plain text interface allows users to inject desired model modifications or qualitative insights - and, as LLMs evolve multimodal capabilities, even non-textual information - while automatically generating fully interpretable, commented constitutive models. While interpretability is not unique among tools such as CKANs or EUCLID, we believe that the CSGA's simplicity and adaptability make it a valuable complement to existing methods. Future work will extend the CSGA evaluation to additional multiaxial loading scenarios, beyond those permitted by our synthetic dataset, and to non-hyperelastic materials. In parallel, we will explore deeper integration of specialized models like the CANN with LLM-based methods such as the SGA, assess performance and robustness gains from more powerful LLM releases, and explore how specialized adaptations of the SGA and similar general LLM-based frameworks can benefit other research domains.

Appendix A: Variability in SGA and CSGA results

We test each SGA and CSGA configuration 25 times and present the best models' predictions above. For transparency, we report the corresponding mean values and standard deviations with Fig. 9. The variations in model performance across the 25 runs of each configuration are substantial: while the standard deviations mostly remain moderate, the best prediction error can be multiple orders of magnitude smaller than the mean. Future work could explore stabilizing the constitutive model generation process by SGA and CSGA to reduce

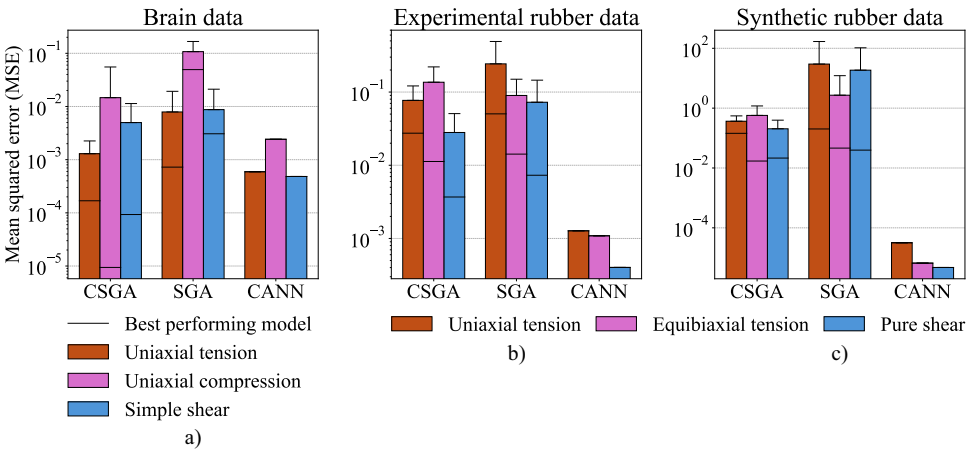

Fig. 9 Report of the best prediction error, mean prediction error, and standard deviation across 25 test runs conducted with the constitutive scientific generative agent (CSGA) and scientific generative agent (SGA) on each dataset. For reference, prediction errors of the constitutive artificial neural network (CANN) tested as a benchmark are also provided

the number of runs and computational cost. Once the random seeds in the neural network within the CANN are fixed, there is no variation in the CANN training and prediction process. We provide the CANN's error values for reference.

Appendix B: Hyperparameter settings

Table 2 Summary of the common hyperparameter configurations used during the fitting of all three constitutive modeling approaches (CANN, SGA and CSGA) to brain tissue data, synthetic rubber data and experimental rubber data

Common hyperparameter settings	
Optimizer	Adam
Learning rate	0.001
Loss	Mean Squared Error

Table 3 Hyperparameter configurations of the constitutive artificial neural network (CANN) used for tests on the brain tissue deformation dataset, the experimental rubber dataset, and the synthetic rubber dataset

CANN hyperparameter settings			
	Brain	Experimental rubber	Synthetic rubber
Hidden layers	1	2	2
Neurons per hidden layer	100	8	8
Epochs	5000	4000	4000

Table 4 Hyperparameter configurations, including the best performing large language models (LLMs), of the scientific generative agent (SGA) and the constitutive scientific generative agent (CSGA) used for tests on the brain tissue deformation dataset, the experimental rubber dataset, and the synthetic rubber dataset

SGA/CSGA hyperparameter settings			
	Brain	Experimental rubber	Synthetic rubber
SGA: best LLM	o1-preview	o3-mini	o1-preview
CSGA: best LLM	o3-mini	o3-mini	o1-preview
Epochs	1000	1000	1000

Appendix C: CSGA model code

C.1 CSGA model for synthetic rubber dataset

```
import torch


class Physics(torch.nn.Module):

    def __init__(self):
        """
        Define trainable continuous physical parameters for differentiable
        optimization.
        Tentatively initialize the parameters with default values.
        """
        super().__init__()

        self.params = [
            # Define the physical parameters as torch.nn.Parameter objects and
              strictly keep them in
            # the list self.params. Do not unpack this list. Define as many
              parameters as list
            # elements as necessary. For each element, replace [float] with the
              desired default value.

            torch.nn.Parameter(torch.tensor([0.1], requires_grad=True)),  # C1
            torch.nn.Parameter(torch.tensor([0.1], requires_grad=True))   # C2
        ]

    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute First Piola-Kirchhoff stress tensor from deformation gradient
        tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            first_piola_kirchhoff_stress (torch.Tensor): First Piola-Kirchhoff
            stress tensor (B, 3, 3).
        """
        # Unpack material parameters
        C1 = self.params[0]  # (1,)
        C2 = self.params[1]  # (1,)

        # Batch size
        B = F.shape[0]  # ()

        # Compute Right Cauchy-Green deformation tensor C = F^T F
        C_tensor = torch.matmul(F.transpose(1, 2), F)  # (B, 3, 3)

        # Compute invariants I1 and I2
        I1 = C_tensor.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1, 1)
        # (B, 1, 1)
        C_tensor_squared = torch.matmul(C_tensor, C_tensor)  # (B, 3, 3)
        I2 = 0.5 * (I1.view(-1) ** 2 - C_tensor_squared.diagonal(dim1=1, dim2=2)
            .sum(dim=1)).view(-1, 1, 1)  # (B, 1, 1)

        # Compute derivative of W with respect to C
        # For incompressible Mooney-Rivlin material:
        # W/C = C1 * I + C2 * (I1 * I - C)
        identity = torch.eye(3, device=F.device).unsqueeze(0).expand(B, 3, 3)
        # (B, 3, 3)
        dWdC = C1 * identity + C2 * (I1 * identity - C_tensor)  # (B, 3, 3)
```

```
        # Compute First Piola-Kirchhoff stress tensor P = 2 * F * dW/dC
        P = 2 * torch.matmul(F, dWdC)  # (B, 3, 3)

        # Ensure P is zero when F is identity
        F_identity = identity  # (B, 3, 3)
        C_identity = torch.matmul(F_identity.transpose(1, 2), F_identity)
        # (B, 3, 3)
        I1_identity = C_identity.diagonal(dim1=1, dim2=2).sum(dim=1).view(-1, 1,
    1)
        # (B, 1, 1)
        C_identity_squared = torch.matmul(C_identity, C_identity)  # (B, 3, 3)
        I2_identity = 0.5 * (I1_identity.view(-1) ** 2 - C_identity_squared
            .diagonal(dim1=1, dim2=2).sum(dim=1)).view(-1, 1, 1)  # (B, 1, 1)
        dWdC_identity = C1 * identity + C2 * (I1_identity * identity -
   C_identity)
        # (B, 3, 3)
        P_identity = 2 * torch.matmul(F_identity, dWdC_identity)  # (B, 3, 3)

        # Adjust P to ensure zero stress at zero strain
        P = P - P_identity  # (B, 3, 3)

        return P
```

C.2 CSGA model for brain dataset

```
import torch

class Physics(torch.nn.Module):

    def __init__(self):
        """
        Define trainable continuous physical parameters for differentiable
        optimization.
        Tension (x >= 0): f(x) = mu_t * x + a_t * x^3, with default mu_t=3.5 and
        a_t=50.0.
        Compression (x < 0): f(x) = mu_c * x + a_c * x^3, with default mu_c=5.27
        and a_c=631.0.
        Shear (off-diagonals): f_shear = mu_s * F_shear + a_s * (F_shear)^3,
   with
        default mu_s=0.782 and a_s=46.9.
        """
        super().__init__()

        # Tension parameters (for principal stretches, x = S - 1 >= 0)
        self.mu_t = torch.nn.Parameter(torch.tensor([3.5], requires_grad=True))
        # (1,) tension modulus
        self.a_t  = torch.nn.Parameter(torch.tensor([50.0], requires_grad=True))
        # (1,) tension cubic coefficient

        # Compression parameters (for principal stretches, x = S - 1 < 0)
        self.mu_c = torch.nn.Parameter(torch.tensor([5.27], requires_grad=True))
        # (1,) compression modulus
        self.a_c  = torch.nn.Parameter(torch.tensor([631.0], requires_grad=True)
   )
        # (1,) compression cubic coefficient

        # Shear parameters (for off-diagonals)
        self.mu_s = torch.nn.Parameter(torch.tensor([0.782], requires_grad=True)
   )
        # (1,) shear modulus
        self.a_s  = torch.nn.Parameter(torch.tensor([46.9], requires_grad=True))
        # (1,) shear cubic coefficient
```

```
        # Store all physical parameters in a list as required (do not unpack
    this
        # list)
        self.params = [
            self.mu_t,
            self.a_t,
            self.mu_c,
            self.a_c,
            self.mu_s,
            self.a_s
        ]


    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute First Piola-Kirchhoff stress tensor from deformation gradient
        tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            first_piola_kirchhoff_stress (torch.Tensor): First Piola-Kirchhoff
            stress tensor (B, 3, 3).
        """
        # Compute SVD of F: F = U * diag(S) * V^T
        # U: (B, 3, 3), S: (B, 3), Vh: (B, 3, 3)
        U, S, Vh = torch.linalg.svd(F, full_matrices=True)
        # U: (B, 3, 3), S: (B, 3), Vh: (B, 3, 3)
        V = Vh.transpose(-2, -1)                                # V: (B, 3, 3)

        # Compute the deviation of principal stretches: x = S - 1, shape: (B, 3)
        x = S - 1.0                                             # (B, 3)

        # Apply piecewise constitutive law for principal stretches:
        # Tension (x >= 0): f(x) = mu_t * x + a_t * x^3
        # Compression (x < 0): f(x) = mu_c * x + a_c * x^3
        f = torch.where(
            x >= 0,
            self.mu_t * x + self.a_t * (x ** 3),            # (B, 3)
            self.mu_c * x + self.a_c * (x ** 3)             # (B, 3)
        )

        # Create a diagonal tensor from f in the principal basis: (B, 3, 3)
        diag_f = torch.diag_embed(f)                            # (B, 3, 3)

        # Reconstruct the full principal stress tensor:
        # P_principal = U * diag_f * V^T, shape: (B, 3, 3)
        P_principal = torch.matmul(torch.matmul(U, diag_f), V.transpose(-2, -1))
        # (B, 3, 3)

        # To decouple normal and shear contributions, extract only the diagonal
        # part:
        # P_principal_diag contains stress only on the principal directions
        # (B, 3, 3)
        P_principal_diag = torch.diag_embed(torch.diagonal(P_principal, dim1=1,
            dim2=2))  # (B, 3, 3)

        # Compute the shear (off-diagonal) contribution:
        # Extract the diagonal of F and subtract to focus on off-diagonals.
        F_diag = torch.diag_embed(torch.diagonal(F, dim1=1, dim2=2))  # (B, 3,
    3)
        F_shear = F - F_diag                                                # (B, 3,
    3)

        # Apply the shear constitutive law element-wise:
```

```
        # P_shear = mu_s * F_shear + a_s * (F_shear)^3, shape: (B, 3, 3)
        P_shear = self.mu_s * F_shear + self.a_s * (F_shear ** 3)     # (B, 3,
    3)

        # Total First Piola-Kirchhoff stress: sum of principal (diagonal-only)
    and
        # shear contributions.
        first_piola_kirchhoff_stress = P_principal_diag + P_shear     # (B, 3,
    3)
        return first_piola_kirchhoff_stress
```

Appendix D: Prompts

D.1 SGA prompt

```
## Scenario:

You are an intelligent AI assistant for coding, physical simulation, and
    scientific
discovery. Follow the user's requirements carefully and make sure you understand
them. Your expertise is strictly limited to physical simulation, material
    science,
mathematics, and coding. Keep your answers short and to the point.
Do not provide any information that is not requested. Always document your code
    as
comments to explain the reason behind them. Use Markdown to format your solution
    .
You are very familiar with Python and PyTorch. Do not use any external libraries
other than the libraries used in the examples.


## Task Requirements
1. Your task is to model the constitutive behavior of a material: in each
iteration, implement a PyTorch module that computes the First
Piola-Kirchhoff stress tensor P from the deformation gradient tensor F.
2. The material is isotropic and incompressible. Feel free to experiment with
different and even non physical constitutive models. The constitutive
behavior searched for is non-linear.


## Format Requirements

### PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of
    dimensions
match before the operation. For example, when multiplying 'J' (B,) and 'I'
(B, 3, 3), you should do 'J.view(-1, 1, 1)' before the operation. Similarly,
'(J - 1)' should also be reshaped to '(J - 1).view(-1, 1, 1)'. If you are not
sure, write down every component in the expression one by one and annotate its
dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use 'A.diagonal(dim1=1,
dim2=2).sum(dim=1).view(-1, 1, 1)'. Avoid using 'torch.trace' or 'Tensor.trace'
since they only support 2D matrix.

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For
example, '# (B, 3, 3)'.
3. The only library allowed is PyTorch. Follow the examples provided by the user
and check the PyTorch documentation to learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with
differentiable optimization and the symbolic constitutive law represented by
PyTorch code. Define them respectively in the '__init__' function and the
'forward' function. Keep the continuous physical parameters in the list
```

```
'self.params'.
5. The output of the 'forward' function is the First Piola-Kirchhoff stress
   tensor
P.
6. The proposed code should strictly follow the structure and function
   signatures
below:

```python
import torch


class Physics(torch.nn.Module):

    def __init__(self):
        """
        Define trainable continuous physical parameters for differentiable
        optimization.
        Tentatively initialize the parameters with default values.
        """
        super().__init__()

        self.params = [
            """
            Define the physical parameters as torch.nn.Parameter objects and
            strictly keep them in the list 'self.params'. Do not unpack this
            list. Define as many parameters as list elements as necessary.
            For each element, replace [float] with the desired default value.

            torch.nn.Parameter(torch.tensor([float], requires_grad=True))
            ...
            """
        ]


    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute First Piola Kirchhoff stress tensor from deformation gradient
        tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            first_piola_kirchhoff_stress (torch.Tensor): First Piola Kirchhoff
            stress tensor (B, 3, 3).
        """
        return first_piola_kirchhoff_stress
```

Solution Requirements

1. Analyze step-by-step what the potential problem is in the previous iterations
based on the feedback. Think about why the results from previous constitutive
laws mismatched with the ground truth. Do not give advice about how to optimize.
Focus on the formulation of the constitutive law. Start this section with
"### Analysis". Analyze all iterations individually, and start the subsection
for each iteration with "#### Iteration N", where N stands for the index.
Remember to analyze every iteration in the history.
2. Think step-by-step what you need to do in this iteration. Think about
how to separate your algorithm into a continuous physical parameter part
and a symbolic constitutive law part. Describe your plan in pseudo-code,
written out in great detail. Remember to update the default values of the
trainable physical parameters based on previous optimizations. Start
this section with "### Step-by-Step Plan".
3. Output the code in a single code block "'''python ... '''" with detailed
```

```
comments in the code block. Do not add any trailing comments before or after
the code block. Start this section with "### Code".
```

D.2 CSGA prompt

```
## Scenario:

You are an intelligent AI assistant for coding, physical simulation, and
    scientific
discovery. Follow the user's requirements carefully and make sure you understand
them. Your expertise is strictly limited to physical simulation, material
    science,
mathematics, and coding. Keep your answers short and to the point.
Do not provide any information that is not requested. Always document your code
    as
comments to explain the reason behind them. Use Markdown to format your solution
    .
You are very familiar with Python and PyTorch. Do not use any external libraries
other than the libraries used in the examples.


## Task Requirements
1. Your task is to model the constitutive behavior of a material: in each
iteration, implement a PyTorch module that computes the First
Piola-Kirchhoff stress tensor P from the deformation gradient tensor F.
2. The material is isotropic and incompressible. Feel free to experiment with
different and even non physical constitutive models. The constitutive
behavior searched for is non-linear.


## Constitutive behavior to be captured:
### Tension:
Deformation Gradient Tensor [component 1,1], First Piola-Kirchhoff Stress Tensor
[component 1,1]
1.0938,0.3650
1.0875,0.3125
1.0750,0.2366
1.0562,0.1661
1.0625,0.1856
1.0188,0.0666
1.1000,0.4151
1.0312,0.1010
1.0500,0.1488
1.0688,0.2091
1.0813,0.2710
1.0250,0.0838
1.0437,0.1324
### Compression:
Deformation Gradient Tensor [component 1,1], First Piola-Kirchhoff Stress Tensor
[component 1,1]
0.9938,-0.0308
0.9688,-0.1908
0.9875,-0.0659
0.9500,-0.3504
0.9812,-0.1040
0.9125,-0.8837
0.9563,-0.2920
1.0000,0.0000
0.9250,-0.6579
0.9437,-0.4127
0.9625,-0.2375
0.9750,-0.1479
0.9187,-0.7630
0.9375,-0.4866
### Simple Shear:
```

```
Deformation Gradient Tensor [component 1,2], First Piola-Kirchhoff Stress Tensor
[component 1,2]
0.1875,0.4557
0.1375,0.2292
0.1750,0.3791
0.1000,0.1412
0.1625,0.3227
0.0250,0.0294
0.1125,0.1649
0.2000,0.5435
0.0500,0.0633
0.0875,0.1186
0.1250,0.1942
0.1500,0.2698
0.0375,0.0486
0.0750,0.0983


## Format Requirements

### PyTorch Tips
1. When element-wise multiplying two matrix, make sure their number of
   dimensions
match before the operation. For example, when multiplying 'J' (B,) and 'I'
(B, 3, 3), you should do 'J.view(-1, 1, 1)' before the operation. Similarly,
'(J - 1)' should also be reshaped to '(J - 1).view(-1, 1, 1)'. If you are not
sure, write down every component in the expression one by one and annotate its
dimension in the comment for verification.
2. When computing the trace of a tensor A (B, 3, 3), use 'A.diagonal(dim1=1,
dim2=2).sum(dim=1).view(-1, 1, 1)'. Avoid using 'torch.trace' or 'Tensor.trace'
since they only support 2D matrix.

### Code Requirements

1. The programming language is always python.
2. Annotate the size of the tensor as comment after each tensor operation. For
example, '# (B, 3, 3)'.
3. The only library allowed is PyTorch. Follow the examples provided by the user
and check the PyTorch documentation to learn how to use PyTorch.
4. Separate the code into continuous physical parameters that can be tuned with
differentiable optimization and the symbolic constitutive law represented by
PyTorch code. Define them respectively in the '__init__' function and the
'forward' function. Keep the continuous physical parameters in the list
'self.params'.
5. The output of the 'forward' function is the First Piola-Kirchhoff stress
   tensor
P.
6. The proposed code should strictly follow the structure and function
   signatures
below:

'''python
import torch


class Physics(torch.nn.Module):

    def __init__(self):
        """
        Define trainable continuous physical parameters for differentiable
        optimization.
        Tentatively initialize the parameters with default values.
        """
        super().__init__()

        self.params = [
```

```
            """
            Define the physical parameters as torch.nn.Parameter objects and
            strictly keep them in the list 'self.params'. Do not unpack this
            list. Define as many parameters as list elements as necessary.
            For each element, replace [float] with the desired default value.

            torch.nn.Parameter(torch.tensor([float], requires_grad=True))
            ...
            """
        ]


    def forward(self, F: torch.Tensor) -> torch.Tensor:
        """
        Compute First Piola Kirchhoff stress tensor from deformation gradient
        tensor.

        Args:
            F (torch.Tensor): deformation gradient tensor (B, 3, 3).

        Returns:
            first_piola_kirchhoff_stress (torch.Tensor): First Piola Kirchhoff
            stress tensor (B, 3, 3).
        """
        return first_piola_kirchhoff_stress
```

```
### Solution Requirements

1. Try to model the constitutive behavior with principal stretches. This appears
    to
be the most promising approach.
2. When there is no strain, indicated by a deformation gradient value of 1,
the first Piola-Kirchhoff stress tensor must be zero.
3. Analyze step-by-step what the potential problem is in the previous iterations
based on the feedback. Think about why the results from previous constitutive
laws mismatched with the ground truth. Do not give advice about how to optimize.
Focus on the formulation of the constitutive law. Start this section with
"### Analysis". Analyze all iterations individually, and start the subsection
for each iteration with "#### Iteration N", where N stands for the index.
Remember to analyze every iteration in the history.
4. Think step-by-step what you need to do in this iteration. Think about how to
separate your algorithm into a continuous physical parameter part and a symbolic
constitutive law part. Describe your plan in pseudo-code, written out in great
detail. Remember to update the default values of the trainable physical
parameters based on previous optimizations. Start this section with
"### Step-by-Step Plan".
5. Output the code in a single code block "'''python ... ''' " with detailed
comments in the code block. Do not add any trailing comments before or after
the code block. Start this section with "### Code".
```

Acknowledgements M.B. gratefully acknowledges funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Projektnummer 535656357.

Author Contributions M.T. conceptualized the study, developed the methodology, created the software, validated the results, conducted formal analysis and investigation, wrote the original draft, reviewed and edited the manuscript, and created the visualizations. M.B. contributed to the conceptualization and methodology, carried out the investigation, and reviewed and edited the manuscript. K.B. was involved in the conceptualization and reviewed and edited the manuscript. K.A. validated the results, curated the data, and reviewed and edited the manuscript. K.L. contributed to the conceptualization, reviewed and edited the manuscript, and supervised the project. C.C. reviewed and edited the manuscript, supervised the project, and acquired funding. R.A. was involved in the conceptualization, reviewed and edited the manuscript, created the visualizations, supervised the project, and managed the project administration.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability As the CANN has already been tested on all three datasets used in this study, they are publicly available at our institute's CANN repository at https://github.com/ConstitutiveANN/CANN and Stanford University's Living Matter Lab's CANN repository at https://github.com/LivingMatterLab/CANN.

Code Availability The code of the CANN is available at our institute's CANN repository at https://github.com/ConstitutiveANN/CANN and Stanford University's Living Matter Lab's CANN repository at https://github.com/LivingMatterLab/CANN. The code of the CSGA is available at https://github.com/ConstitutiveSGA/CSGA.

Declarations

Competing Interests The authors declare no competing interests.

Ethical Approval No animals or human subjects were used in this study. All data are taken from the literature: [39].

References

1. Mooney M. A theory of large elastic deformation. J Appl Phys. 1940;11(9):582–92.
2. Rivlin R. Large elastic deformations of isotropic materials. i. fundamental concepts. Philos Trans A Math Phys Eng Sci. 1948;240(822):459–90.
3. Rivlin RS. Large elastic deformations of isotropic materials iv. further developments of the general theory. Philos Trans A Math Phys Eng Sci. 1948;241(835):379–97.
4. Ogden RW. Large deformation isotropic elasticity-on the correlation of theory and experiment for incompressible rubberlike solids. Proc R Soc Lond A Math Phys Sci. 1972;326(1567):565–84.
5. Kirchdoerfer T, Ortiz M. Data-driven computational mechanics. Comput Methods Appl Mech Eng. 2016;304:81–101.
6. Carrara P, De Lorenzis L, Stainier L, Ortiz M. Data-driven fracture mechanics. Comput Methods Appl Mech Eng. 2020;372:113390.
7. Ghaboussi J, Garrett J Jr, Wu X. Knowledge-based modeling of material behavior with neural networks. J Eng Mech. 1991;117(1):132–53.
8. Hashash Y, Jung S, Ghaboussi J. Numerical implementation of a neural network based material model in finite element analysis. Int J Numer Meth Eng. 2004;59(7):989–1005.
9. Sussman T, Bathe K-J. A model of incompressible isotropic hyperelastic material behavior using spline interpolations of tension-compression test data. Commun Numer Methods Eng. 2009;25(1):53–63.
10. Latorre M, Montáns FJ. Extension of the sussman-bathe spline-based hyperelastic model to incompressible transversely isotropic materials. Comput Struct. 2013;122:13–26.
11. Dal H, Denli FA, Açan AK, Kaliske M. Data-driven hyperelasticity, part i: a canonical isotropic formulation for rubberlike materials. J Mech Phys Solids. 2023;179:105381.
12. Fuhg JN, Anantha Padmanabha G, Bouklas N, Bahmani B, Sun W, Vlassis NN, Flaschel M, Carrara P, De Lorenzis L. A review on data-driven constitutive laws for solids. Arch Comput Methods Eng. 2024; 1–43.
13. Hao Z., Liu S., Zhang Y., Ying C., Feng Y., Su H., Zhu J.: Physics-informed machine learning: A survey on problems, methods and applications. arXiv:2211.08064 (2022)
14. Linden L, Klein DK, Kalina KA, Brummund J, Weeger O, Kästner M. Neural networks meet hyperelasticity: a guide to enforcing physics. J Mech Phys Solids. 2023;179:105363.
15. As' ad F, Avery P, Farhat C. A mechanics-informed artificial neural network approach in data-driven constitutive modeling. Int J Numer Method Eng 2022;123(12), 2738–2759
16. Linka K, Hillgärtner M, Abdolazizi KP, Aydin RC, Itskov M, Cyron CJ. Constitutive artificial neural networks: a fast and general approach to predictive data-driven constitutive modeling by deep learning. J Comput Phys. 2021;429:110010.
17. Linka K, Pierre SRS, Kuhl E. Automated model discovery for human brain using constitutive artificial neural networks. Acta Biomater. 2023;160:134–51.
18. McCulloch JA, St Pierre SR, Linka K, Kuhl E. On sparse regression, lp-regularization, and automated model discovery. Int J Numer Method Eng. 2024;125(14):7481.
19. Abdolazizi KP, Linka K, Cyron CJ. Viscoelastic constitutive artificial neural networks (vcanns)-a framework for data-driven anisotropic nonlinear finite viscoelasticity. J Comput Phys. 2024;499:112704.
20. Tushar KS, Chakraborty S. Fusion-based constitutive model (fuce): toward model-data augmentation in constitutive modeling. Int J Mech Syst Dyn.
21. Taç V, Rausch MK, Costabal FS, Tepole AB. Data-driven anisotropic finite viscoelasticity using neural ordinary differential equations. Comput Methods Appl Mech Eng. 2023;411:116046.
22. Koza JR. Genetic programming: On the programming of computers by means of natural selection (complex adaptive systems). Bradford Book. 1993;1:18.
23. Abdusalamov R, Hillgärtner M, Itskov M. Automatic generation of interpretable hyperelastic material models by symbolic regression. Int J Numer Meth Eng. 2023;124(9):2093–104.

24. Flaschel M, Kumar S, De Lorenzis L. Unsupervised discovery of interpretable hyperelastic constitutive laws. Comput Methods Appl Mech Eng. 2021;381:113852.
25. Flaschel M, Kumar S, De Lorenzis L. Discovering plasticity models without stress data. npj Comput Mater 2022;8(1):91.
26. Joshi A, Thakolkaran P, Zheng Y, Escande M, Flaschel M, De Lorenzis L, Kumar S. Bayesian-euclid: Discovering hyperelastic material laws with uncertainties. Comput Methods Appl Mech Eng. 2022;398:115225.
27. Thakolkaran P, Joshi A, Zheng Y, Flaschel M, De Lorenzis L, Kumar S. Nn-euclid: Deep-learning hyperelasticity without stress data. J Mech Phys Solids. 2022;169:105076.
28. Kolmogorov AN. On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables. Am Math Soc. 1961
29. Liu Z, Wang Y, Vaidya S, Ruehle F, Halverson J, Soljačić M, Hou TY, Tegmark M. Kan: Kolmogorov-arnold networks. 2024. arXiv:2404.19756
30. Liu Z, Ma P, Wang Y, Matusik W, Tegmark M: Kan 2.0: Kolmogorov-arnold networks meet science. 2024. arXiv:2408.10205
31. Abdolazizi KP, Aydin RC, Cyron CJ, Linka K. Constitutive kolmogorov-arnold networks (ckans): Combining accuracy and interpretability in data-driven material modeling. 2025. arXiv:2502.05682
32. Han Z, Jiang W. Synthetic data enhances mathematical reasoning of language models based on artificial intelligence. Inf Technol Control. 2025;54(1):345–58.
33. Chen A, Stanton SD, Alberstein RG, Watkins AM, Bonneau R, Gligorijevic V, Cho K, Frey NC. LLMs are highly-constrained biophysical sequence optimizers. In: NeurIPS 2024 Workshop on AI for New Drug Modalities. 2024.
34. Ma P, Wang T-H, Guo M, Sun Z, Tenenbaum JB, Rus D, Gan C, Matusik W. LLM and simulation as bilevel optimizers: a new paradigm to advance physical scientific discovery. In: Forty-first International conference on machine learning. 2024.
35. Pandey S, Xu R, Wang W, Chu X. Openfoamgpt: a rag-augmented llm agent for openfoam-based computational fluid dynamics. arXiv:2501.06327 (2025)
36. Zhang B, Ma H, Ding J, Wang J, Xu B, Lin H. Distilling implicit multimodal knowledge into large language models for zero-resource dialogue generation. Inf Fusion. 2025;102985.
37. Holzapfel GA. Nonlinear solid mechanics: a continuum approach for engineering science. Kluwer Academic Publishers Dordrecht. 2002.
38. Pierre SRS, Linka K, Kuhl E. Principal-stretch-based constitutive neural networks autonomously discover a subclass of ogden models for human brain tissue. Brain Multiphys. 2023;4:100066.
39. Budday S, Sommer G, Birkl C, Langkammer C, Haybaeck J, Kohnert J, Bauer M, Paulsen F, Steinmann P, Kuhl E, et al. Mechanical characterization of human brain tissue. Acta Biomater. 2017;48:319–40.
40. Budday S, Sommer G, Haybaeck J, Steinmann P, Holzapfel GA, Kuhl E. Rheological characterization of human brain tissue. Acta Biomater. 2017;60:315–29.
41. Budday S, Ovaert TC, Holzapfel GA, Steinmann P, Kuhl E. Fifty shades of brain: a review on the mechanical testing and modeling of brain tissue. Arch Comput Methods Eng. 2020;27:1187–230.
42. Treloar LR. Stress-strain data for vulcanized rubber under various types of deformation. Rubber Chem Technol. 1944;17(4):813–25.
43. Steinmann P, Hossain M, Possart G. Hyperelastic models for rubber-like materials: consistent tangent operators and suitability for treloar's data. Arch Appl Mech. 2012;82:1183–217.
44. Han Y, Duan J, Wang S. Benchmark problems of hyper-elasticity analysis in evaluation of fem. Materials. 2020;13(4):885.
45. Bien-aimé LKM, Blaise BB, Beda T. Characterization of hyperelastic deformation behavior of rubber-like materials. SN Appl Sci. 2020;2(4):648.
46. Treloar LG. The physics of rubber elasticity. 1975.
47. Kolmogorov AN. On the representations of continuous functions of many variables by superposition of continuous functions of one variable and addition. Dokl Akad Nauk USSR. 1957;114:953–6.