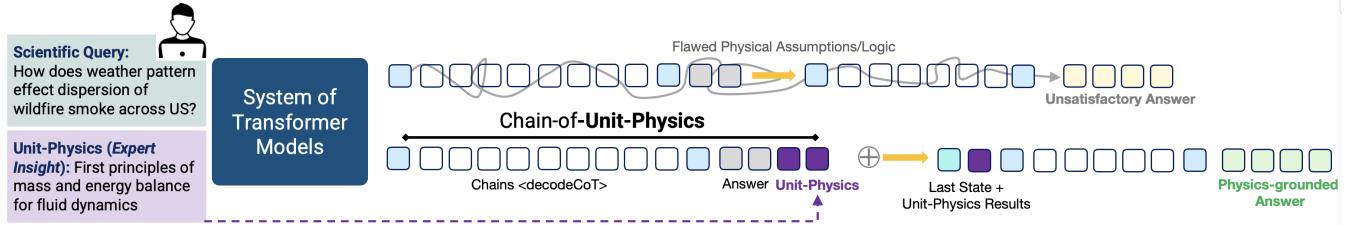


Chain of Unit-Physics: A Primitive-Centric Approach to Scientific Code Synthesis

Vansh Sharma^a, Venkat Raman^a

^aUniversity of Michigan, Ann Arbor, 48109-2102, MI, USA

Abstract



Agentic large language models are proposed as autonomous code generators for scientific computing, yet their reliability in high-stakes problems remains unclear. Developing computational scientific software from natural-language queries remains challenging broadly due to (a) sparse representation of domain codes during training and (b) the limited feasibility of RLHF with a small expert community. To address these limitations, this work conceptualizes an inverse approach to code design, embodied in the Chain of Unit-Physics framework: a first-principles (or primitives)-centric, multi-agent system in which human expert knowledge is encoded as unit-physics tests that explicitly constrain code generation. The framework is evaluated on a nontrivial combustion task (12 degrees-of-freedom), used here as a representative benchmark for scientific problem with realistic physical constraints. Closed-weight systems and code-focused agentic variants fail to produce correct end-to-end solvers, despite tool and web access, exhibiting four recurrent error classes: interface (syntax/API) hallucinations, overconfident assumptions, numerical/physical incoherence, and configuration fragility. Open-weight models with chain-of-thought (CoT) decoding reduce interface errors but still yield incorrect solutions. On the benchmark task, the proposed framework converges within 5–6 iterations, matches the human-expert implementation (mean error of $3.1 \times 10^{-3}\%$), with a ~33.4% faster runtime and a ~30% efficient memory usage at a cost comparable to mid-sized commercial APIs, yielding a practical template for physics-grounded scientific code generation. As datasets and models evolve, zero-shot code accuracy will improve; however, the Chain of Unit-Physics framework goes further by embedding first-principles analysis that is foundational to scientific codes, thereby guiding more reliable and interpretable human–AI collaboration.

Keywords: Large Reasoning Models (LRM), Scientific Code Generation, First-principles, Primitives, Combustion, Chain-of-Thought (CoT), Agentic AI

1. Introduction

Emergence of large language models (LLMs) as key drivers of the integration of artificial intelligence (AI) across multiple domains, including computational science [1, 2, 3], have enabled intuitive human-like interactions with complex systems that outperform previous machine learning (ML) methods [4]. However, these models frequently exhibit unwarranted confidence, producing fluent but potentially erroneous outputs [5], a liability that has serious implications in high-stakes engineering domains such as aircraft and engine design. In particular, during code generation scenarios [6, 7], LLMs might generate programs with correct syntactic structure [8] that may still violate logical requirements or drift from specified constraints over extended iterations [9]. Such limitations underscore the urgent need for rigorous verification frameworks to ensure the accuracy and reliability of LLM-driven workflows.

From an engineering perspective, scientific code generation

tasks diverge fundamentally from traditional software development. General purpose applications, such as mobile apps or utility libraries, prioritize rapid iteration, user experience, and flexibility, often tolerating minor glitches and relying on integration tests. In contrast, scientific software demands mathematically rigorous algorithms, bit-for-bit reproducibility, and exhaustive validation against analytical or reference benchmarks [10]. Moreover, it must be optimized for large-scale numerical workloads on high-performance computing platforms without sacrificing stability [11], leveraging precision-oriented languages and parallel libraries (e.g. MPI [12]) to ensure both performance and accuracy. Given these divergent workflows and the pivotal role of testing in ensuring correctness, current research on LLM-driven code generation emphasizes on developing test cases from existing codebases [13, 14, 15] and transforming them into formal, verifiable unit-test specifications [16]. This also highlights that validation currently re-

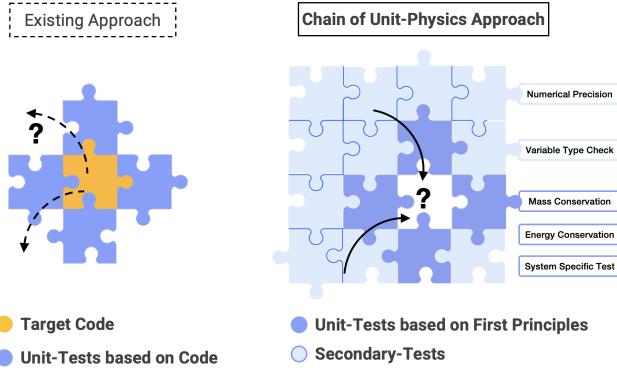


Figure 1: Conceptual difference between Chain of Unit-Physics approach and existing methods. Left: the existing “code-first” approach, where unit tests are written after implementation, merely exposing latent errors and forcing rework. Right: the proposed approach, in which a human expert specifies first-principles unit tests (e.g., conservation laws) that guide code generation.

lies on comparing code outputs with predefined input-output datasets, an approach that is infeasible when problems are complex or due to limited number of experts in the field. In such domains, agents often generate erroneous codes that require extensive human debugging. In addition, for a given problem, there could be multiple correct code implementations, complicating validation based solely on output matching. While it may seem intuitive that providing unit tests alongside problem specifications would automatically improve the accuracy of LLM-generated code, this assumption has not been rigorously validated [17]. The true effect of such test suites on the fidelity and robustness of model output remains largely unexplored [18], and, in particular, the mechanisms by which LLMs interpret, apply, and iteratively refine their code based on these tests [19] have yet to be systematically investigated.

Generating unit tests from existing code can inadvertently perpetuate latent errors [20], particularly when those tests are not limited to generic checks, such as data type or format tests, rather emphasize algorithmic correctness [21]. This issue is especially acute in the development of custom computational fluid dynamics (CFD) solvers [22, 23] built on libraries such as OpenFOAM [24] and AMReX [25]. To overcome these limitations, this work systematically applies an inverse code design methodology (see Fig. 1), formally known as test-driven development (TDD) [26], to scientific software in combustion science, one such domain where TDD approaches have not been thoroughly evaluated. This work proposes Chain of Unit-Physics, an approach that embeds human expert knowledge directly into distinct reasoning chains of the agentic system via “unit-physics”—formalized, testable constraints that encode fundamental physics (e.g., energy conservation laws) and practitioner experience (e.g., dimensional / bound checks, and floating-point diagnostics). This inverse-design method provides two key advantages in scientific software. First, human-authored tests embed deep domain expertise (first principles or primitives) into targeted validation checks, ensuring that each algorithmic component faithfully represents the underlying physics. Second, because these verification suites are au-

thored by specialists, they impose stringent quality criteria – any failure of the LLM generated code then clearly indicates a gap in the expert’s test specification itself, thereby shifting the source of error away from the model and onto the test suite or expert’s own knowledge. This discussion naturally prompts the question: *“If we can so precisely formalize our requirements, why not use a single numerical solver across all combustion applications, simply adapting for different fuel phases?”*. While a universal solver for all combustion problems in this context might seem appealing, it is neither practical nor conducive to the advancement of scientific discovery. There are a myriad of approaches to solving complex set of equations, including the choice of models, the choice of numerical schemes, their implementation as well as the specific regime of validity for all these choices. Instead, designing targeted “unit-physics” tests or “primitives”, each grounded in first principles, provides a more direct, transparent, and reliable framework for developing and verifying solver components for specific physical processes. Beyond the framework, a key contribution of this study is the systematic evaluation of human-authored unit tests to recast fundamental physics checks as stringent formal specifications for AI-driven code generation, thus reducing model ambiguity and minimizing error propagation. The following sections describe the proposed framework in detail (§ 2) and evaluate it on a benchmark scientific task (§ 3).

2. Methodology

2.1. Multi-Agent System

AI systems with higher degree of autonomy and goal-directed behavior, with a notion of planning and reasoning [27], embedded during model training, are termed as “agents” in this framework. While this definition is evolving, current work uses open-weight instruction-tuned decoder-only LLMs as the base for agents: Llama 3.3 70B Instruct [28] and GPT-OSS-20B model with o200k_harmony tokenizer [29]. Based on additional experiments (not shown), the framework adheres to established guidelines for the selection of model parameters (e.g. sampling temperature), specified in the respective reference articles [28, 29]. The models are accessed using the Transformers library [30] and vLLM library [31] while the agents are orchestrated through custom code within a Python 3.11 environment. Inference is performed with PyTorch [32] (GPU-accelerated where available), while task decomposition and prompt management use custom code abstractions. The framework runs inside a dedicated Python virtual sandbox environment that is not pre-configured with metadata on all available libraries. The agent is granted isolated code execution privileges within this sandbox, ensuring that any dependency installs or script executions cannot affect the system root directory or global files. During code execution, any missing dependencies are detected using builtin the subprocess library. Configuration and runtime states are managed with the logging library, and human-expert guidance can be integrated via command line interface (CLI).

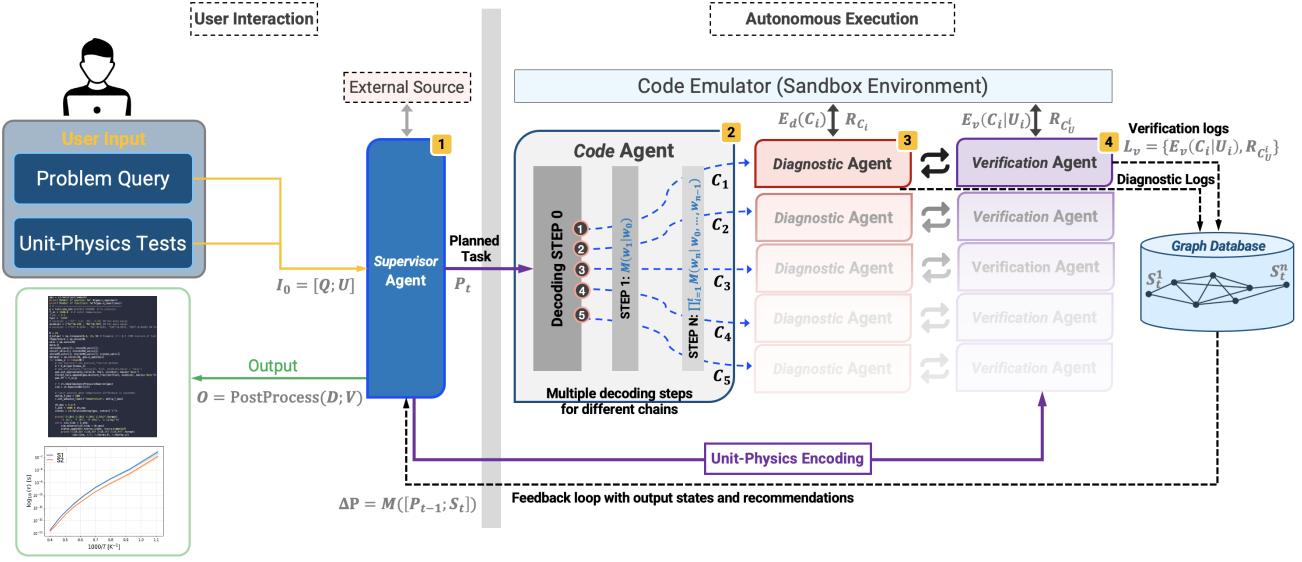


Figure 2: Chain of Unit-Physics workflow: User queries and unit-physics tests are processed by a supervisor agent (1), which orchestrates chain-of-thought (CoT) code generation (2); diagnostic (3) and verification (4) agents then evaluate the code against physics-based tests and expert knowledge, feeding back signals that steer code synthesis toward physically consistent solutions. The final green-bordered block confirms successful query execution.

Figure 2 illustrates a multi-agent LLM-driven scientific workflow that transforms a domain-specific user query into actionable code and visual results. In the input, the user’s scientific question is paired with ‘basis prompts’ that establish execution permissions, tool availability, coding language settings, and transfer protocols. Additional input scopes the unit-physics tests that concatenated with the scientific query to form the complete input to the framework. The Supervisor Agent is the only part of the framework that interacts with the User. This agent identifies the objective by extracting critical parameters (e.g., thermodynamic variables, software/language requirements) and generates a structured plan that sequences the necessary steps and tool invocations and then sets up an autonomous iterative loop: each task is passed to Code Agent, which uses a custom decoding algorithm [33] that elicits intermediate reasoning traces. The modified decoding algorithm branches only at the first generated token: instead of taking the greedy top-1 token, it spawns multiple paths by taking each of the top-k candidates, then continues each path with standard greedy or beam search decoding. This procedure generates multiple code candidates per task similar to Chain-of-Thought (CoT) [34], without relying on explicit “think step by step” instructions. For every completed path, the algorithm identifies the answer span and computes a “confidence” score as the average margin between the probabilities of the top-1 and the top-2 tokens over those answer tokens. These chains can be evaluated in parallel, either on separate GPUs or, depending on model memory requirements, jointly on a single GPU. While these self-assigned confidence scores enable chain selection via a user-specified confidence threshold, they are not a substitute for validation against first-principles physical constraints and numerical consistency checks.

The pruned candidate chains are then asynchronously routed to two downstream agents, a Diagnostic agent and a Verifica-

tion agent, which together constitute “chain of unit-physics”. The Diagnostic Agent first performs preliminary checks (e.g. dependency installation) by automatically running code in the emulator, analyzing any errors, and applying targeted corrections. Subsequently, the Verification Agent applies formalized unit-physics tests to assess physical and numerical consistency. These primitives yield physics-grounded verification even without reference datasets. At each stage of the framework, the agent states and logs for different interactions persist in a graph-based database. The database stores the summarized execution and diagnostic logs for each code candidate on nodes, to avoid context-window limitations [5], while the edges define the logic route and transitions. If any candidate code cannot be corrected within a user-specified number of attempts, the complete error log and the candidate implementation are returned to the Supervisor Agent for updating the plan as needed, optionally incorporating human feedback. Finally, the agent consolidates the output into domain-relevant visualizations (for example, line graphs and contour graphs of key quantities of interest), closing the loop between natural-language inquiry and quantitative scientific insight.

The proposed framework is formally described as follows:
let M be the decoder-only Transformer model with $Q = \text{Tokenize}(\text{UserQuery})$ and $U = \text{Tokenize}(\text{UnitPhysicsTests})$ concatenated into the initial supervisor input $I_0 = [Q; U]$. The sandboxed execution operator $\text{Exec}(\cdot)$ returns a pair (R, E) where R is the runtime output (or \emptyset on failure or no valid output) and E is the error message (or \emptyset if successful or no error message). Then the workflow is:

- **Supervisor Initialization.**

$$P_0 = M(I_0) \quad (1)$$

- **Multi-Chain Code Decoding (Code Agent).** For iter-

ations $t = 1, 2, \dots$ the supervisor produces a decoding prompt P_{t-1} and the code agent generates K candidate programs via multi-chain decoding:

$$\{C_t^{(k)}\}_{k=1}^K = \text{CoTDecode}_K(M, P_{t-1}) \quad (2)$$

where each chain k branches at the first token.

- **Code Execution.** For each candidate chain k :

$$(R_t^{(k)}, E_t^{(k)}) = \text{Exec}(C_t^{(k)}) \quad (3)$$

- **Diagnostics (Diagnostic Agent).**

$$D_t^{(k)} = \text{Diag}(C_t^{(k)}, R_t^{(k)}, E_t^{(k)}) \quad (4)$$

- **Testing Against Unit-Physics (Verification Agent).**

$$(V_t^{(k)}, \hat{E}_t^{(k)}) = \text{Verify}(C_t^{(k)}, U) \quad (5)$$

- **Logging & State Aggregation (Database).**

$$L_d^{(t)} = \{(C_t^{(k)}, R_t^{(k)}, D_t^{(k)})\}_{k=1}^K, \quad L_v^{(t)} = \{(C_t^{(k)}, V_t^{(k)}, \hat{E}_t^{(k)})\}_{k=1}^K \quad (6)$$

$$S_t = \text{Summarize}(L_d^{(t)}, L_v^{(t)}) \quad (7)$$

- **Supervisor Plan Update.** With optional external guidance H_t :

$$\Delta P_t = M([P_{t-1}; S_t; H_t]), \quad P_t = \text{Refine}(P_{t-1}, \Delta P_t) \quad (8)$$

- **Termination & Final Output.** If some chain k^* satisfies diagnostic and verification criteria:

$$k^* = \arg \max_k \text{Score}(D_t^{(k)}, V_t^{(k)}), \quad O = \text{PostProcess}(R_t^{(k^*)}) \quad (9)$$

otherwise, continue with iteration $t + 1$.

Notation:

- $\text{CoTDecode}_K(M, P)$: multi-chain decoding that branches on the first token and greedily completes K candidate programs under prompt P .
- Diag : diagnostic analysis of runtime behavior and errors.
- Verify : execution of candidate code against unit-physics tests U .
- Summarize : compression of execution and verification logs into a state S_t .
- $\text{Refine}(P, \Delta P)$: update of the supervisor plan.
- Score : ranking function combining diagnostic and verification signals.
- $\text{PostProcess}(R)$: formatting, data manipulation, and visualization of the selected result.

2.2. Unit-Physics Encoding

Unit-physics primitives encode the first-principles constraints that a combustion expert would impose on any admissible thermochemical state. These constraints may be problem-specific (e.g., detonations vs. constant-pressure reactors) or generic to reacting mixtures, and are designed to be portable across mechanisms and numerical implementations (*primitives are domain-portable*). In contrast to conventional unit tests, which mostly compare implementation details against reference outputs, primitives enforce conservation, thermodynamic consistency, and physical consistency, and thus provide partial verification even when no ground-truth solution is available.

For example, in a problem consisting of a spatially homogeneous (zero-dimensional) reactor that integrates species mass fractions Y_i , temperature T , pressure p , and density ρ , typical primitives include:

- **Species mass conservation:** $|\sum_i Y_i - 1| \leq \epsilon$, with $\epsilon = 10^{-16}$.
- **Equation-of-state residual for an ideal mixture:** $|p - \rho RT/\bar{W}| \leq \epsilon$, where \bar{W} is the molecular weight of the mixture.
- **Physical bounds:** $1 \geq Y_i \geq 0$, $T_{\min} \leq T \leq T_{\max}$, $p > 0$, $\rho > 0$.
- **Conservation of inert diluents (e.g. nitrogen):** $|Y_{N_2}^{(1)} - Y_{N_2}^{(2)}| \leq \epsilon$ between two states of the same closed system.
- **Dimensional consistency of thermochemical quantities:** h_i in J kg^{-1} , $\dot{\omega}_i$ in $\text{kg m}^{-3} \text{s}^{-1}$ for species enthalpy and mass production rate.

Similarly, for a one-dimensional Zeldovich-von Neumann-Döring (ZND) detonation structure [35, 36, 37], additional process-specific primitives supplement these generic checks. Denoting pre- and post-shock states by superscripts ⁽¹⁾ and ⁽²⁾ and $v = 1/\rho$ as specific volume, we impose:

- **Rankine-Hugoniot energy closure:** $|h_2 - h_1 + \frac{1}{2}(p_2 + p_1)(v_2 - v_1)| \leq \epsilon$.
- **Chapman-Jouguet (CJ) condition at the equilibrium product state:** $\text{Mach}_{\text{CJ}} = 1$.
- **Consistency of product states between ZND and equilibrium (HP) calculations:** $\left| \frac{T^{\text{ZND}} - T^{\text{HP}}}{T^{\text{HP}}} \right| \leq \epsilon$ and $|\chi_i^{\text{ZND}} - \chi_i^{\text{HP}}| \leq \epsilon$ for major species mass or mole fractions χ_i .

These primitives can be enforced during both synthesis and evaluation of agent-generated codes, providing thermochemically grounded verification signals even in the absence of curated reference datasets. Within the AI framework, they may be supplied either as natural-language descriptions or as a structured JSON specification that is parsed into executable checks.

3. Results

This study will consider a canonical reactor-design problem (prompt shown below) to calculate the ignition delay time

(IDT) for two different types of systems: 1. Closed-weight models and 2. Open-weight models. Although the Cantera library [38] provides built-in routines to perform this calculation, the system is deliberately prompted to implement the time integrators directly for the full $N + 1$ degree-of-freedom (dof) problem (12 dofs for H₂), rather than calling these high-level functions. The fundamental nature of this task makes it well suited to illustrate why unit-physics must be explicitly encoded in the system, and the same lesson extends to analogous problems in other scientific domains. Since this is a domain-specific realistic problem with nontrivial numerical constraints and the available benchmarks do not resolve such fine-grained combustion physics or numerical aspects, there is limited information on the model behavior here. The present case therefore serves as a proxy for real engineering problems that fall outside standard benchmark coverage. The closed-weight models are evaluated using their APIs, and open-weight models are distributed on 4 NVIDIA H100 GPUs based on workload.

Prompt for Zero-D Reactor Task

You need to develop an ignition delay calculator for different fuels. They key here is to provide an explicit Euler or RK scheme to integrate states and using Cantera lib in python. STRICT: You cannot use Cantera's reactor functions. You can only obtain gas object related properties from Cantera. For starters you can use fuel as hydrogen at temperature of 1300 Kelvin at pressure of 101325 Pa for stoichiometric composition.

3.1. Closed-weight Models and Systems

First, closed-weight models such as ChatGPT[39], Claude Sonnet 4.5 [40] and Gemini 2.5 Pro [41] are tested for the given reactor prompt. All models have CoT, have access to local Python emulator to verify the codes¹ and can access the Web during script generation. The results in the upper half of Table 1 show that none of the evaluated models produced the target code, and detailed analysis highlights an incorrect understanding of ignition delay time, however, which does not impede code execution.

OpenAI ChatGPT: The function `derivatives()` that computes derivatives of state variables with time fails on execution. The code shown in the snippet exemplifies an API hallucination: it calls a nonexistent `gas-object` method (reported as `int_enegeries_mass`), which is a clear syntax/API error based on Cantera documentation. An additional fundamental error lies in the process assumption: the model treats the evolution as a constant-pressure process rather than the intended constant-volume process. Furthermore, the model confidently suggests using `gri30.yaml` mechanism (CH₄ focused) for H₂ combustion as it will contain relevant reactions.

¹Codes to be released after paper is accepted.

Code snippet from ChatGPT: `derivatives()`

```
... # mass fractions
Y = s_massdens / rho
# update gas state so Cantera returns consistent properties
gas.TPY = T, P, Y # T [K], P [Pa], mass fractions
# molar net production rates [kmol/m3/s], shape (n_species,)
omega_dot = gas.net_production_rates # kmol / m3 / s
# molecular weights [kg/kmol]
W = gas.molecular_weights # kg / kmol
# mass production rates [kg/m3/s]
mdot = W * omega_dot # elementwise, kg/m3/s
# specific internal energies of species [J/kg]
u_species = gas.int_energies_mass ← Error
# mixture constant-volume specific heat [J/kg/K]
cv_mix = gas.cv_mass
# dT/dt from internal energy balance (closed homogeneous system)
numerator = np.dot(u_species, mdot) # J/m3/s
dTdt = - numerator / (rho * cv_mix) ... ← Error
```

Anthropic Claude Sonnet: The model yields the most elaborate deliverables: an algorithmic data-flow outline, a report, and an instructions file, even though such artifacts were not requested in the prompt. This likely reflects internal prompt expansion or meta-scaffolding [42], where a smaller model expands the query with detailed problem specifications to reduce hallucinations. The model selects the correct reaction mechanism (`h2o2.yaml`), but the final code implementation was incorrect: during RK time integration, it produces a negative temperature, indicating numerical/physical instability. After an expert review of the code and additional tests with smaller time steps, a second issue (similar to ChatGPT’s code) was identified: the gas state was being set using an incompatible combination of thermodynamic formulation. The code snippet shows a function that is formulate by the model to update states for a constant volume system, but it currently applies a constant-pressure energy formulation which is highly incorrect. Instead of using species enthalpies (h_k) and c_p , it should use species internal energies and mixture c_v . Thus, even if the RK step had succeeded, the incorrect state-setting logic would have caused a subsequent failure.

Code	snippet	from	Sonnet:
	<code>get_derivatives_const_vol()</code>		

```
...# set gas state: temperature, density, mass fractions
gas.TDY = T, rho, Y # T [K], ρ [kg/m3], Y mass fractions
# molar net production rates [kmol/m3/s]
omega = gas.net_production_rates
# molecular weights [kg/kmol]
W = gas.molecular_weights
# mass-fraction time derivatives [1/s]
dYdt = omega * W / rho # elementwise multiply
# species specific enthalpies [J/kg]
h = gas.partial_molar_enthalpies / W ← Error
# mixture constant-pressure specific heat [J/kg/K]
cp = gas.cp_mass
# dT/dt from enthalpy balance (closed homogeneous system)
dTdt = - np.dot(h, omega * W) / (rho * cp) ... ← Error
```

Google Gemini: In comparison to other models, Gemini did capture the correct process (constant-volume) assumption but fails at input initialization, requesting a mechanism file, `h2_gri30.yaml`, that does not exist. Upon manually fix-

Table 1: Results for different closed-weight models and agents. The possible outcomes are: ✓ indicates successful output, ▶ represents partially successful output, and ✗ signifies a failure.

Model	Output	Failed Step	Evaluation
ChatGPT	CanteraError: gas object has no attribute 'int_energies_mass'	Derivative	✗
Sonnet 4.5	CanteraError: Phase::setTemperature T = -56199.3659	RK integration	✗
Gemini 2.5 Pro	CanteraError: h2_gri30.yaml not found	Incorrect inputs	✗
AI System	Output	Failed Step	Evaluation
Codex v0.44.0	CanteraError: gas object has no attribute 'enthalpies_mass'	Derivative	✗
Claude Code v2.0.5	CanteraError: Phase::setTemperature T is negative	RK integration	✗

ing this error, the code still shows erroneous outputs: shown in the snippet is a condition that masks a problem - if Cantera raises a state error, the code will allow the solver to continue and silently provide misleading output after that point. Non-experts using these models to write scientific code need extreme caution: models may make fundamentally wrong physical assumptions and mask them, even if they follow the provided documentation.

```
Code snippet from Gemini: calculate_derivatives()

... try:
gas.TPY = T, P, Y
except Exception as e:
# Handle cases where the state is invalid (e.g., T < 0)
print(f"Error setting Cantera state: {e}")
# Return zeros to avoid crashing the solver
return np.zeros_like(y_state) ... ← Error
```

Since the focus is on AI systems, we also evaluate agentic deployments of these models (see Table 1). The agentic variants: Codex [6] and Claude Code [43] exhibit the same failure patterns observed in their base models - given that the agent orchestrations are layered on the same underlying models despite having access to Web. The Claude Code agent ignores the instruction to implement an Euler integrator, instead providing only an RK4 scheme with faulty adaptive time-stepping logic and the same thermodynamic inconsistency observed for Sonnet 4.5. Additionally, the Codex agent repeats the same syntactic mistake as the GPT model. The agents systems are able to navigate and work with files in local directories, creating requested files in the correct locations, but still end up generating erroneous code. Essentially, workflow or agentic system does not remediate core model/API errors as observed previously. Furthermore, across models and systems, we observe a systematic mechanism mis-specification: the code tends to default to `gri30.yaml` even for H₂ fuel, which undermines fidelity from the outset. Despite being trained by different companies, the models select mechanisms based on what is most documented and frequently co-occurs with terms such as "Cantera" and "ignition," not on physical fidelity, so they all tend to converge on the same overrepresented mechanism in their training data distribution.

Collectively, these outcomes illustrate four recurring error classes in agentic code synthesis for scientific workflows: (i) interface hallucinations (nonexistent methods/attributes), (ii)

over-assumption about scientific process (hard assumptions that do not translate to correct codes), (iii) numerical and physical incoherence (instabilities such as negative T and misuse of thermodynamic state variables), and (iv) configuration fragility (missing files and unsuitable default mechanisms).

3.2. Open-weight Models and Systems

Open-weight models include the Llama (sampling temperature of 0.6) and OSS models (sampling temperature of 0.3 and reasoning effort set to medium), and are accessed using the framework described in §2. Two types of tests are performed here: first with the standard model and second with model coupled to CoT decoding algorithm to improve program search efficiency. During both these tests, the models do not have access to the Web or a python emulator. From top row of Table 2, across all models, the synthesized codes² fail on basic physics/API checks. Llama-3.3-70B produces NaN states during the RK4 step, indicating numerical/physical instability during the integration process. Adding CoT does not fully resolve such errors; while output code executes, but the model instead mis-defines IDT (defined by the model as the time when T≥T+100), a conceptual or semantic error that runs the code but produces the wrong output. OSS-20B without CoT already fails at input handling and state initialization: it invokes an invalid Cantera call and overwrites the gas state in two incorrect ways: (i) treating mole fractions as mass fractions and (ii) calling the wrong function for setting for mass fractions. With CoT enabled, the failure mode shifts to inconsistent thermodynamics: the model implicitly treats the scalar `gas.enthalpy_mass` as a species-resolved vector, producing a temperature derivative of the wrong shape (see code snippet). Overall, CoT shifts the dominant failure mode from calling nonexistent functions to misusing valid APIs—an improvement, but still insufficient. Hence, one CoT-based model is marked 'partially successful': the codes executed without any API errors, but the IDT compute logic was incorrect. We also note the same mechanism mis-specification (use of `gri30.yaml` for H₂ fuel) in these tests as well.

²Codes to be released after paper is accepted.

Table 2: Results for different open-weight models and agent. See Table 1 for outcome description.

Model	Output	Failed Step	Evaluation
Llama3.3-70B	Error in RK4 function: Not-a-Number (NaN) outputs	RK integration	✗
Llama3.3-70B + CoT	Incorrect definition of IDT (time where $\text{gas.T} \geq 100 + T_0$)	IDT logic	✗
OSS-20B	CanterError: syntax 'set_mass_fractions' and gas.T is not writeable	Inputs & RK	✗
OSS-20B + CoT	Error: $d\text{Tdt}$ wrong size. Occurs as gas.enthalpy_mass is assumed for each specie	Incorrect logic	✗
AI System	Output	Failed Step	Evaluation
CoT+Unit Phy (AI+Human)	Ignition delay time: 1.13179e-05 s	-	✓

Code snippet from OSS-20B (no CoT): `init()`

```
... # create gas with GRI-Mech 3.0 (H2/O2/N2 chemistry)
gas = ct.Solution('gri30.yaml')
# set initial thermodynamic state: T [K], P [Pa]
gas.TP = 1300.0, 101325.0 # 1300 K, 1 atm
# stoichiometric H2/air mixture: H2 + 0.5 O2 + 3.76 N2
# unnormalized "masses" proportional to m_H2 = 1.0, m_O2 = 0.5,
m_N2 = 3.76
masses = np.array([1.0, 0.5, 3.76])
# normalize to sum to 1 (convert to mass fractions)
masses /= masses.sum()
# set mass fractions in Cantera
gas.set_mass_fractions(masses) ... ← Error
```

Code snippet from OSS-20B (CoT): `rhs()`

```
... T = y[0]
X = y[1:]
gas.TPX = T, gas.P, X
rho = gas.density
cv = gas.cp_mass - ct.gas_constant / gas.mean_molecular_weight
h = gas.enthalpy_mass ← Error
w = gas.net_production_rates
W = gas.molecular_weights
dTdt = - (1.0 / (rho * cv)) * np.dot(h, w)
dXdt = (1.0 / rho) * (w / W) ...
```

3.2.1. Chain of Unit-Physics System

The primitives-based AI system successfully synthesized the correct solution. Llama-3.3-70B model (sampling temperature of 0.6) supervised planning and tool use, while task-specific coding agents were instantiated with OSS-20B (sampling temperature of 0.3 and reasoning effort set to medium) for its code-generation proficiency. The system operated on a retrieval budget (at most two web queries), limiting dependence on external search and consistent with other evaluations in this study. Figure 3 summarizes the workflow as a state graph focusing on the solution and not agent blocks, where each node (colored circle) is a state or CoT step annotated with its confidence score (if available). Following the method in §2, the initial query is passed to the supervisor agent that performs targeted fact/equation checks, formulates a plan, and communicates specific coding instructions to a code agent that explores four parallel candidates (one per GPU). The CoT states (in Fig. 3) record progressive fixes across candidates—for example, correcting the Cantera mechanism format from `.cti` to `.yaml`. Candidates with confidence below 0.4 (user-defined) are pruned (gray

states) and remaining candidates are passed to the Diagnostic agent. This agent performs an initial sanity check by executing the code and resolving issues relating to dependencies before proceeding, if needed. In our setup, the Python environment intentionally did not include Cantera. On execution, the agent correctly captured the traceback `ModuleNotFoundError`: No module named 'cantera'. As a diagnostic repair, it issued a single command, `! pip install cantera`, to resolve the error³. Once the Diagnostic agent checks a candidate, the Verification Agent enforces the human-defined unit-physics tests. One primitive initially failed due to a mis-specified reaction mechanism (as shown). This failure is logged and stored in the graph database, and then fed back to the Supervisor Agent, which updates the plan and triggers another iteration. The system requires roughly 5-6 iterations to synthesize code that satisfies all checks. After these corrections, the primitive-grounded code runs successfully, yielding an IDT of 1.13179e-05 s. For statistical insights, this test case is repeated five times, obtaining four successful runs and one case where the system exhausts its iteration budget before converging.

Code Performance

The code produced by the framework is evaluated against a reference implementation developed by a human expert and compared in three dimensions: (1) execution time, (2) memory usage, and (3) L² error. The representative chemical conditions with $\phi = 1$ and $p = 1$ atm with H₂-O₂ combustion and numerical conditions of $dt = 1e^{-10}$ with RK4 integrator are fixed and only the input temperature is varied from 1300 to 2400 K.

Figure 4 compares the performance of the proposed framework (green) with a reference code developed by a human expert (orange). In terms of runtime [plot (a)], the human-expert code consistently achieves a longer time to solution—on the order of 32–34s (33.4% on average) slower across the temperature range—indicating that the framework implementation is more optimized for wall-clock performance. The performance of the AI code can be attributed to using vectorized energy evaluations instead of explicitly looping over each species. Additionally, the proposed framework is more memory efficient [plot (b)], reducing peak memory usage from roughly 270 MB for the human code to about 200 MB (reduction of nearly 30%), with only weak dependence on temperature. The accuracy of the generated solver is quantified by the L² error between the

³Execution privileges are explicitly granted for these runs; such commands should be used with caution.

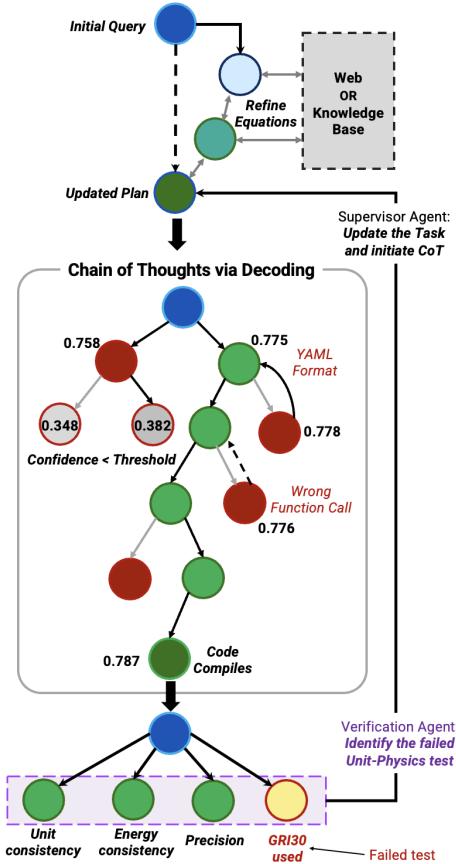


Figure 3: Approximate states of the reactor task with primitives: (●): correct state, (○): input to agent, (○): mismatch detected, (○): pruned state and (●): incorrect state. Numbers are CoT-confidence score self-reported by the Code agent.

two solutions [plot (c)]. The error remains below 10^{-4} for all tested temperatures (mean relative error of $3.1 \times 10^{-3}\%$), with absolute match for some temperatures and a modest increase at higher temperatures, demonstrating that Chain of Unit-Physics closely reproduces the human-expert solution while trading a small increase in runtime for a substantial reduction in memory usage. Upon additional code review, the improved memory footprint of Chain of Unit-Physics arises from the way the AI-generated code organizes data: state variables are packed into a single contiguous structure rather than being split across multiple arrays and objects, which reduces overhead and allocator fragmentation. The slight slowdown in runtime was expected due to additional safeguard introduced by the model, an internal high temperature-bounds check ($T \geq 4000$ K) that is frequently evaluated during the integration, however the vectorized approach offsets the time lost in checks. This extra validation step improves robustness, but adds a small computational penalty (~ 5 s) relative to human-optimized implementation.

Cost Analysis

The economic competitiveness and cost profile of the proposed framework are summarized in Table 3. For the reference reactor task, Codex v0.44.0 consumes 352297 input and 16098

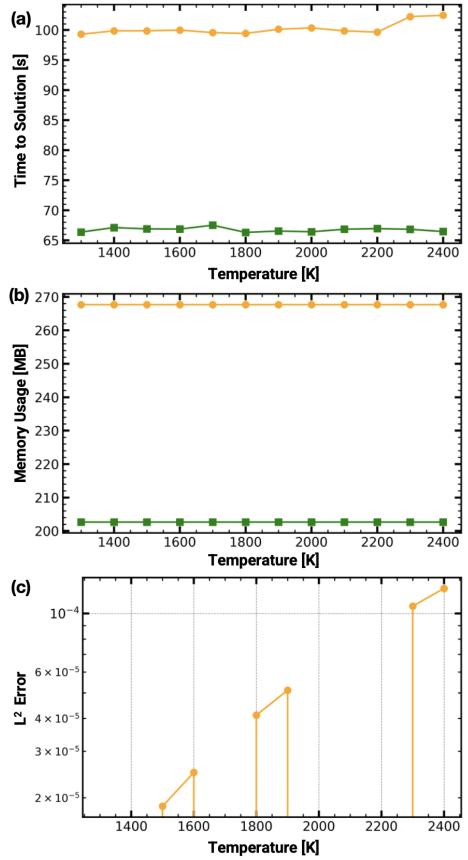


Figure 4: Comparison between: (●): Human+AI (Chain of Unit-Physics), and (○): Human, developed code implementation. Plots show (a) Time to solution,(b) Peak memory usage and (c) L² error of the Chain of Unit-Physics solution with respect to the human-expert reference (only single curve).

output tokens at a total API cost of \$0.25, while Claude Code v2.0.5 uses substantially fewer tokens (32110 / 2796) and therefore incurs a lower absolute cost of \$0.07. Normalizing by token count, however, reveals that Claude Code is actually more expensive on a per-token basis than Codex, so its apparent advantage is due primarily to greater token efficiency rather than cheaper pricing. In contrast, the proposed Chain-of-Unit-Physics framework processes 225502 input and 25382 output tokens on local GPUs, yielding a comparable or larger token budget than the commercial systems. The agent-level average token usage further indicates that most of this budget is concentrated in the Supervisor, Code, and Verification agents, with the Diagnostic agent contributing relatively less to the usage. These trends are consistent with the intended roles of the agents: the Supervisor oversees the entire process and therefore consumes the largest share of tokens, while the Code and Verification agents repeatedly iterate over the code once the Diagnostic agent has resolved dependency issues during the initial iterations.

To further contextualize the API cost of the proposed framework relative to commercial providers, Fig. 5 breaks down the projected cost for different model variants. For the fixed Chain-of-Unit-Physics workload, the resulting API cost spans more than two orders of magnitude across providers. Among Ope-

Table 3: Token usage and cost for different systems and agents.

AI System	Input Tokens	Output Tokens	Total Cost [\\$]
Codex v0.44.0	352 297	16 098	0.25
Claude Code v2.0.5	32 110	2 796	0.07
Chain of Unit-Physics^a	225 502	25 382	-
↳ Supervisor Agent	22 050	1 466	-
↳ Code Agent	8 840	556	-
↳ Diagnostic Agent	1 435	661	-
↳ Verification Agent	8 365	2 101	-

^a Agent token count is averaged over successful runs.

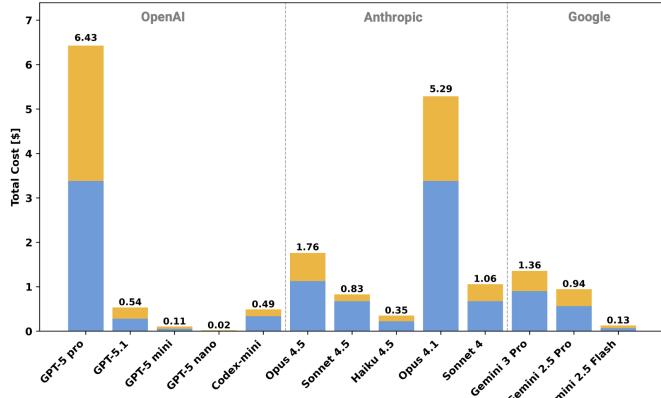


Figure 5: Overall cost of the reactor task for different providers: (●): output token cost, (○): input token cost. Numbers are based on pricing guides for respective providers as of Nov. 2025.

nAI and Google models, lightweight variants such as GPT-5 nano and Gemini 2.5 Flash are the most economical, at approximately \\$0.02 and \\$0.13 per run, respectively, while mid-sized models (GPT-5 mini, Codex-mini, Sonnet 4.5, Haiku 4.5, Gemini 2.5 Pro) fall in the \\$0.1–\\$1 range. In contrast, frontier chat models such as GPT-5 pro and Anthropic Opus are substantially more expensive, at \\$6.43 and \\$5.29 for the same token budget. Since Chain-of-Unit-Physics is implemented with mid-sized models, its effective cost is in the same regime as the mid-size hosted APIs, while remaining up to two to three orders of magnitude cheaper than the highest-end commercial offerings. Moreover, the Codex and Claude Code systems did not produce fully correct solutions for this task, so any practical deployment of those systems would likely require multiple retries or additional verification steps, further increasing their effective cost. Collectively, these observations suggest that the proposed framework is cost-competitive with commercial baselines for this workload, and that further gains in cost-effectiveness might come from techniques such as KV-cache compression [44] or batch prompting [45] than from fundamental changes to the overall concept.

4. Conclusion

This work examined ignition-delay computation for hydrogen combustion as a representative, high-stakes scientific task for agentic code generation. Closed-weight systems equipped

with web access failed to produce a correct end-to-end solution. The experiments reveal four recurring error classes in agentic scientific coding: (i) interface hallucinations (nonexistent methods or attributes), (ii) overconfident assumptions about the scientific process (hard-wired logic that does not implement the correct algorithm), (iii) numerical and physical incoherence (e.g. invalid thermodynamic states), and (iv) configuration fragility (missing files or unsuitable default mechanisms). Across providers, models tended to select the same reaction mechanism, driven by its prominence in online documentation and frequent co-occurrence with “Cantera” and “ignition,” rather than any explicit consideration of physical fidelity. Open-weight models display the same pattern: CoT decoding reduces interface hallucinations, but mainly shifts errors toward misuse of valid APIs.

The proposed Chain of Unit-Physics framework attains a correct solution in 5 to 6 iterations for the same ignition-delay problem; measured across five independent runs, four converged, and the remaining failure was attributable to an externally imposed token budget rather than to a modeling error. Performance analysis shows that the generated code closely matches the human-expert reference (L^2 error below 10^{-4}), with approximately 33.4% faster runtime and using about 30% less memory, a gain attributable to more compact data handling. When the same token budget is priced under different provider tariffs, the framework places in the cost band of mid-sized hosted models (on the order of \\$0.1–\\$1 per run), while avoiding excessive retries and verification passes that might be necessary for closed-model-based agents.

Overall, the results indicate that current agentic systems, even with tools and web access, are not yet reliable for this class of scientific workflow, and that embedding expert-designed unit-physics primitives as constraints is an effective way to improve reliability without sacrificing economic plausibility. The central idea is the use of portable physics and numerical primitives as an organizing scaffold for code search, rather than any specific underlying model family. This primitives-centric design offers a new horizon for human–AI collaboration in scientific computing: expert constraints define the admissible solution space, models search within that space, and resulting failures remain interpretable enough to train subsequent models. Future research should quantify how unit-test relaxation affects search and incorporate iterative refinement of the unit-physics tests *in situ*, allowing the tests themselves to evolve within the code-generation cycle as new failure modes are discovered. Another direction for future work is to systematically evaluate influence of sampling temperature, an aspect not explored in the present study.

Acknowledgments

This work was supported by the Center for Prediction, Reasoning, and Intelligence for Multiphysics Exploration (C-PRIME), a PSAAP-IV project funded by the Department of Energy, grant number DE-NA0004264 (program manager: Dr. David Etim).

References

- [1] M. Du, Y. Chen, Z. Wang, L. Nie, D. Zhang, Large language models for automatic equation discovery of nonlinear dynamics, *Physics of Fluids* 36 (9) (2024).
- [2] Z. Yang, Y. Bin, Y. Shi, X. I. Yang, Large language model driven development of turbulence models, arXiv preprint arXiv:2505.01681 (2025).
- [3] V. Sharma, V. Raman, A reliable knowledge processing framework for combustion science using foundation models, *Energy and AI* 16 (2024) 100365.
- [4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., Gpt-4 technical report, arXiv preprint arXiv:2303.08774 (2023).
- [5] V. Sharma, V. Raman, A reliable knowledge processing framework for combustion science using foundation models, *Energy and AI* 16 (2024) 100365. doi:<https://doi.org/10.1016/j.egyai.2024.100365>.
- [6] OpenAI, OpenAI Codex (2021). URL <https://github.com/openai/codex>
- [7] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, D. Ha, The ai scientist: Towards fully automated open-ended scientific discovery (2024). arXiv:2408.06292. URL <https://arxiv.org/abs/2408.06292>
- [8] P. Yin, G. Neubig, A syntactic neural model for general-purpose code generation, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2017, pp. 440–450.
- [9] V. Sharma, V. Raman, Steering conceptual bias via transformer latent-subspace activation, arXiv preprint arXiv:2506.18887 (2025).
- [10] V. Raman, M. Hassanaly, Emerging trends in numerical simulations of combustion systems, *Proceedings of the Combustion Institute* 37 (2) (2019) 2073–2089.
- [11] V. Sharma, A. H. Rauch, V. Raman, Accelerating cfd simulations with super-resolution feedback-informed adaptive mesh refinement, in: AIAA SCITECH 2025 Forum, 2025, p. 1467.
- [12] J. J. Dongarra, S. W. Otto, M. Snir, D. Walker, et al., An introduction to the mpi standard, *Communications of the ACM* 18 (11) (1995).
- [13] M. Schäfer, S. Nadi, A. Eghbali, F. Tip, An empirical evaluation of using large language models for automated unit test generation, *IEEE Transactions on Software Engineering* 50 (1) (2023) 85–105.
- [14] C. Paduraru, A. Staicu, A. Stefanescu, Llm-based methods for the creation of unit tests in game development, *Procedia Computer Science* 246 (2024) 2459–2468.
- [15] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, J. Yin, Chatunitest: A framework for llm-based test generation, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 572–576.
- [16] T. Godage, S. Nimishan, S. Vasanthapriyan, V. Palanisamy, C. Joseph, S. Thusethan, Evaluating the effectiveness of large language models in automated unit test generation, in: 2025 5th International Conference on Advanced Research in Computing (ICARC), IEEE, 2025, pp. 1–6.
- [17] W. Wang, H. Ning, G. Zhang, L. Liu, Y. Wang, Rocks coding, not development: A human-centric, experimental evaluation of llm-supported se tasks, *Proceedings of the ACM on Software Engineering 1 (FSE)* (2024) 699–721.
- [18] N. S. Mathews, M. Nagappan, Test-driven development and llm-based code generation, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, 2024, pp. 1583–1594.
- [19] S. Agarwal, P. Nie, M. Nagappan, What inputs drive effective llm-based unit test generation?, *IEEE Software* (2025).
- [20] A. Prasad, E. Stengel-Eskin, J. Chen, Z. Khan, M. Bansal, Learning to generate unit tests for automated debugging, in: Second Conference on Language Modeling, 2025.
- [21] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, E. Wang, Automated unit test improvement using large language models at meta (2024). arXiv:2402.09171. URL <https://arxiv.org/abs/2402.09171>
- [22] S. Sharma, R. Bielawski, O. Gibson, S. Zhang, V. Sharma, A. H. Rauch, J. Singh, S. Abisleiman, M. Ullman, S. Barwey, et al., An amrex-based compressible reacting flow solver for high-speed reacting flows relevant to hypersonic propulsion, arXiv preprint arXiv:2412.00900 (2024).
- [23] S. Zhang, V. Sharma, V. Raman, High-fidelity numerical analyses for hydrogen jet-flames in crossflow, in: Proceedings of the ASME Turbo Expo 2025, American Society of Mechanical Engineers, Tennessee, USA, 2025, paper No. 154178.
- [24] H. Jasak, Openfoam: Open source cfd in research and industry, *International journal of naval architecture and ocean engineering* 1 (2) (2009) 89–94.
- [25] W. Zhang, A. Myers, K. Gott, A. Almgren, J. Bell, Amrex: Block-structured adaptive mesh refinement for multiphysics applications, *The International Journal of High Performance Computing Applications* 35 (6) (2021) 508–526.
- [26] K. Beck, *Test driven development: By example*, Addison-Wesley Professional, 2022.

- [27] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt, A. B. Murthy, Position: Llms can't plan, but can help planning in llm-modulo frameworks, in: Forty-first International Conference on Machine Learning, 2024.
- [28] A. Grattafiori, et. al, The llama 3 herd of models (2024). arXiv:2407.21783. URL <https://arxiv.org/abs/2407.21783>
- [29] S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao, et al., gpt-oss-120b & gpt-oss-20b model card, arXiv preprint arXiv:2508.10925 (2025).
- [30] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, 2020, pp. 38–45.
- [31] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, I. Stoica, Efficient memory management for large language model serving with pagedattention, in: Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 611–626. doi:10.1145/3600006.3613165. URL <https://doi.org/10.1145/3600006.3613165>
- [32] S. Imambi, K. B. Prakash, G. Kanagachidambaresan, Pytorch, Programming with TensorFlow: solution for edge computing applications (2021) 87–104.
- [33] X. Wang, D. Zhou, Chain-of-thought reasoning without prompting, Advances in Neural Information Processing Systems 37 (2024) 66383–66409.
- [34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models (2023). arXiv:2201.11903.
- [35] Y. B. Zel'Dovich, On the theory of the propagation of detonation in gaseous systems, Zh. eksp. teoret. fiz. 10 (1940) 542–568.
- [36] J. von Neumann, Theory of detonation waves, Tech. Rep. ADB967734, INSTITUTE FOR ADVANCED STUDY PRINCETON NJ, accessed: 2025-10-05 (1942). URL <https://apps.dtic.mil/sti/citations/ADB967734>
- [37] W. Döring, G. Burkhard, Contribution to the theory of detonation (translation from german), Technical Report Tech. Report F-TS-1227-IA, Headquarters, Air Materiel Command, Wright-Patterson Air Force Base, english Translation of original German work (1949).
- [38] D. G. Goodwin, H. K. Moffat, I. Schoegl, R. L. Speth, B. W. Weber, Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes, <https://www.cantera.org>, version 3.1.0 (2024). doi:10.5281/zenodo.14455267.
- [39] OpenAI, Chatgpt (chatgpt), large language model developed by OpenAI. (2025).
- [40] Anthropic, Claude (sonnet 4.5), large language model developed by Anthropic. (2025).
- [41] G. DeepMind, Gemini 2.5: Our most intelligent ai model (2025).
- [42] C. Carpineto, G. Romano, A survey of automatic query expansion in information retrieval, Acm Computing Surveys (CSUR) 44 (1) (2012) 1–50.
- [43] Anthropic, Claude code (v2.0.5) (2025).
- [44] Y. Liu, H. Li, Y. Cheng, S. Ray, Y. Huang, Q. Zhang, K. Du, J. Yao, S. Lu, G. Ananthanarayanan, et al., Cachegen: Kv cache compression and streaming for fast large language model serving, in: Proceedings of the ACM SIGCOMM 2024 Conference, 2024, pp. 38–56.
- [45] J. Lin, M. Diesendruck, L. Du, R. Abraham, Batchprompt: Accomplish more with less, in: The Twelfth International Conference on Learning Representations, 2024.