MaterialsPilot: An Execution-Feedback Framework for Generative Design of Complex Atomistic Architectures

Qiaolin Lu[1] **Tongliang Liu**[2†] **Qiang Qu**[2]
Bo Han[3] **Yi Chang**[4] **Chengqi Zhang**[1]

[1]The Hong Kong Polytechnic University [2]The University of Sydney
[3]Hong Kong Baptist University [4]Jilin University

Abstract

Generating precise complex atomistic architectures from natural language specifications is a frontier in AI-driven materials discovery, critical for realizing functional systems like catalytic active sites and hetero-interfaces. While Large Language Models (LLMs) offer a powerful interface for such tasks, mapping abstract semantics to rigorous structural constraints remains unsolved as LLMs inherently face two deficits: first, the lack of quantitative physicochemical priors leads to topologically imprecise outputs; and second, open-loop generation without physical verification fails to resolve intermediate spatial conflicts, causing error propagation to invalidate coupled multi-step assemblies. To bridge this gap, we introduce MaterialsPilot, which formulates generation as iterative code optimization. It incorporates two key mechanisms: (1) Hierarchical Retrieval to bridge domain knowledge gaps by providing precise domain schemas; and (2) a Physics-Aware Feedback Loop to master complex logic by autonomously debugging execution and physical violations. Across four complex material modeling tasks and seven diverse LLM backbones, MaterialsPilot increases the rate of fully successful generations from 22.0% to 66.4%, while also producing structures that better satisfy user-specified constraints and physical plausibility checks. These results establish a model-agnostic framework for physically reliable language-driven atomistic design. Our code is publicly available at: `github.com/QiaolinLu/MaterialsPilot`.

1 Introduction

The integration of artificial intelligence into materials science has catalyzed a paradigm shift, moving the field beyond the traditional regime of property prediction toward the generative design of atomistic structures. While recent advances have demonstrated success in discovering stable bulk crystals[1, 2], the frontier of functional materials design lies in the direct realization of complex atomistic architectures, specifically functional subsystems such as catalytic active sites, grain boundaries, and hetero-interfaces [3, 4, 5]. These architectures govern the critical performance metrics of next-generation technologies, ranging from efficient energy conversion to semiconductor device scaling. To navigate this complexity, Large Language Models (LLMs) have emerged as a transformative interface, offering the unique potential to democratize materials design by translating high-level natural language specifications, which capture human design intent, into rigorous and physically valid atomic coordinates [6, 7, 8]. This capability represents a fundamental frontier in AI-driven science, promising to bridge the gap between abstract semantic constraints and precise structural realization.

†Corresponding author: Tongliang Liu (`tongliang.liu@sydney.edu.au`).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Here, we study natural-language-guided construction, editing, and assembly of atomistic structures, using initial material structures when provided. Our goal is to generate structures that satisfy the requested compositional and structural requirements while maintaining basic physical plausibility.

However, directly mapping abstract linguistic instructions to rigorous structural constraints remains an unsolved challenge, as general-purpose LLMs inherently face two critical deficits in this domain. First, the lack of quantitative physicochemical priors leads to physically invalid outputs. Trained primarily on general corpora, LLMs process atomic specifications as statistical tokens rather than physical entities constrained by conservation laws and Euclidean geometry. Consequently, they frequently fail to enforce rigorous constraints, such as charge neutrality, space group symmetries, and thermodynamic stability, resulting in structures that appear chemically plausible but are fundamentally flawed [9, 10, 11, 12]. Second, open-loop generation without intermediate verification fails to resolve spatial conflicts in coupled assemblies. Constructing functional systems like hetero-interfaces involves spatially interdependent steps, where the validity of the final structure depends strictly on the precision of initial transformations. Standard LLMs generate these sequences autoregressively in a single pass without checking for physical inconsistencies, such as steric clashes. Thus, a minor numerical deviation in an early step propagates unchecked, rendering the final global topology incoherent [13, 14, 12].

To address these challenges, we introduce MaterialsPilot, a framework for iterative synthesis and repair of structure-construction programs. Building on executable program representations [15, 16, 17], the LLM selects and parameterizes materials-modeling operations, while numerical geometry is delegated to deterministic library routines. Code remains an inspectable intermediate representation: execution produces the candidate structure, and runtime errors or detected constraint violations guide subsequent revision. This design enables construction decisions to be checked and repaired under the task requirements. Specifically, MaterialsPilot combines two mechanisms to support domain-grounded construction and iterative repair: First, to bridge the domain knowledge gap, we incorporate a Hierarchical Retrieval-Augmented Generation system. Recognizing that complex materials require structured blueprints rather than flat textual descriptions, our system organizes crystallographic data into functional tiers. By retrieving precise construction schemas, such as rotation matrices for grain boundaries, rather than generic context, we effectively ground the generation process in established crystallographic rules. Second, to master the sequential logic of construction, we introduce a Physics-Aware Execution-Feedback Loop. This mechanism overcomes the inherent fragility of single-pass inference by mimicking the trial-and-error workflow of human experts. It empowers the agent to execute its generated code, analyze physical violations reported by the simulation engine, and iteratively refine the script. Together, these components allow the framework to inspect and repair candidate structures under the requested constraints.

We evaluate MaterialsPilot on a challenging benchmark spanning four distinct material modeling tasks, including point defects, surface adsorption, grain boundaries, and hetero-interfaces. Across seven LLM backbones, the rate of fully successful generations increases from 22.0% with single-turn code generation without retrieval or feedback to 66.4% with the full framework, with particularly strong gains in topologically complex domains. Beyond strict task completion, MaterialsPilot also improves constraint satisfaction and physical plausibility, indicating that the generated structures better follow both user-specified requirements and atomistic validity checks. Furthermore, qualitative analysis reveals the agent's ability to self-heal by converting execution traces and physics-aware diagnostic signals into corrective actions, establishing a robust and physically reliable paradigm for language-driven materials design [16, 18].

2 Background and Related Work

Generative paradigms in materials design. Traditional deep generative models (e.g., CDVAE [2], MatterGen [1]) excel in continuous latent space optimization but lack interpretability and struggle with complex natural language constraints. Conversely, direct text-to-structure LLMs (e.g., CrystaLLM [19], MatLLMSearch [20]) offer linguistic controllability but suffer from geometric hallucinations and syntactic errors. MaterialsPilot bridges this gap by utilizing *executable Python code* as an intermediate representation. This decouples high-level reasoning from numerical calculation, ensuring geometric precision and effectively mitigating the stochastic failures inherent in direct token prediction.

Scientific agents and physics-aware feedback. While autonomous agents like Coscientist [6] and ChemCrow [7] have revolutionized experimental chemistry, they often overlook the topological

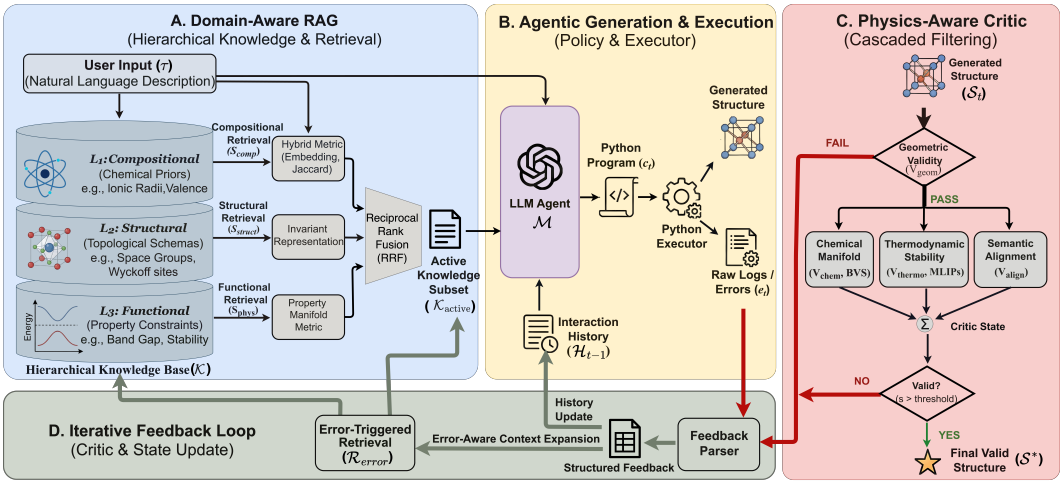

Figure 1: **Overview of MaterialsPilot Framework.** The closed-loop system orchestrates four synergistic modules: (A) Domain-Aware RAG retrieves hierarchical knowledge priors; (B) Agentic Generation & Execution synthesizes and executes Python code; (C) a Physics-Aware Critic computes internal diagnostic states over geometric, chemical, thermodynamic, and semantic constraints; and (D) an Iterative Feedback Loop uses a deterministic feedback parser to verbalize execution traces and critic states into actionable textual feedback for self-correction.

intricacies of periodic crystal construction (e.g., symmetry and interface matching). Furthermore, while recent code-generation benchmarks [14, 21] utilize runtime feedback, they largely focus on syntactic correctness. MaterialsPilot advances this by integrating a *physics-aware* feedback loop. Unlike standard debugging, our framework employs domain-specific validators (e.g., MLIP energy, geometric checks) to guide the agent in refining structures based on thermodynamic viability rather than merely code executability.

Task-aligned evaluation. AtomWorld [10] provides externally defined natural-language structure-manipulation tasks and an official evaluator, with operations such as substitution, movement, rotation, and supercell transformation. AtomWorld-Agent supplies an external baseline aligned with these tasks. Evaluating a frozen MaterialsPilot configuration in this setting tests transfer beyond our four construction domains under independently specified tasks and evaluation rules.

3 Methodology

MaterialsPilot addresses the task defined above through iterative synthesis and repair of executable construction programs. Conditioned on the task input τ, it retrieves domain knowledge, generates a Python program c, executes it to obtain a candidate structure $\mathcal{S}$, and converts runtime errors or detected constraint violations into localized repair feedback. Code is the intermediate representation; the structure is the task output. As shown in Fig. 1, the framework consists of domain-aware retrieval, code-based generation, physics-aware validation, and iterative feedback. Generation-time diagnostics guide repair, while final benchmark performance is measured separately using the frozen post-hoc evaluation protocol.

3.1 Domain-Aware Retrieval-Augmented Generation

We use τ for the complete task input: the natural-language instruction together with any initial structures supplied by the task. These structures form part of the available construction context; they are not assumed to be present for every task. Given τ and a domain knowledge base $\mathcal{K}$, MaterialsPilot first retrieves a task-specific active context and then synthesizes an executable construction program:

$$\mathcal{K}_{\text{active}} = \mathcal{R}(\tau, \mathcal{K}, \mathcal{H}), \qquad c = \mathcal{M}(\tau, \mathcal{K}_{\text{active}}, \mathcal{H}), \tag{1}$$

where $\mathcal{M}$ denotes the LLM, $\mathcal{R}$ is the retrieval function, and $\mathcal{H}$ stores the interaction history from previous execution-feedback rounds. The resulting program c is executed to construct a candidate atomistic structure $\mathcal{S} = \text{Execute}(c)$.

Hierarchical knowledge organization. Materials construction requires heterogeneous knowledge, ranging from element-level chemical rules to topology-specific construction recipes. We therefore

organize $\mathcal{K}$ into three semantic tiers. L_1 contains compositional priors such as elemental heuristics, oxidation-state rules, and radius-based constraints. L_2 contains structural schemas such as symmetry operations, surface templates, defect rules, grain-boundary transformations, and interface construction protocols. L_3 contains functional knowledge, including application-specific preferences such as adsorption configurations, thermodynamic descriptors, and target property constraints. Detailed schemas are provided in Appendix A.

Layer-aligned hybrid retrieval. MaterialsPilot queries each tier using a modality aligned with its semantic content. Compositional retrieval matches elemental similarity and periodic trends in L_1; structural retrieval matches crystallographic or topological similarity in L_2; and functional retrieval matches target property descriptors in L_3. The candidates from these retrieval channels are fused using Reciprocal Rank Fusion:

$$\mathrm{RRF}(d) = \sum_{m \in \{L_1, L_2, L_3\}} \frac{1}{k + \mathrm{rank}_m(d)}, \tag{2}$$

where $\mathrm{rank}_m(d)$ is the rank of document d under modality m and k is a smoothing constant. The top-ranked results are packed into $\mathcal{K}_{\mathrm{active}}$ as structured construction context, including applicable conditions, repair rules, and priority tags. Full definitions of the layer-specific similarity metrics and context packaging strategy are given in Appendix B.

3.2 Agentic Generation & Execution

We formulate atomistic construction as a code-synthesis problem rather than direct coordinate prediction. Conditioned on the task specification τ, the active retrieval context $\mathcal{K}_{\mathrm{active}}$, and previous feedback, the LLM generates a Python script c_t that constructs the target structure through deterministic materials-science operations. The retrieved context provides explicit construction constraints, such as valid chemical species, symmetry operations, surface orientations, interface matching rules, or adsorption-site preferences.

The generated script follows a simple execution contract: it must instantiate an atomistic structure object $\mathcal{S}_t$ and, when available, return construction metadata such as selected sites, transformation matrices, interface spacing, or vacuum thickness. By invoking high-level libraries such as *Pymatgen* [22], the framework delegates numerical geometry operations to deterministic routines, reducing hallucinated coordinates and enforcing periodic-lattice consistency.

The script is then executed in a sandboxed interpreter. Execution produces either a candidate structure $\mathcal{S}_t$ with associated metadata, or an error trace e_t containing syntax, API, or construction-level failures. These execution traces provide grounded evidence for later refinement, while physically implausible but executable structures are further analyzed by the physics-aware critic in Sec. 3.3.

3.3 Physics-Aware Critic for Iterative Refinement

While code execution can detect syntactic or API-level failures, it does not by itself determine whether an executable structure is physically meaningful or consistent with the task specification. We therefore introduce a physics-aware critic $\mathcal{C}(\mathcal{S}, \tau)$ that analyzes each candidate structure $\mathcal{S}$ under task context τ and returns diagnostic signals for iterative refinement. Importantly, this critic is used only during generation; benchmark success is computed separately using the fixed post-hoc evaluation protocol in Sec. 4.2. The critic consists of four complementary checks: **geometric validity**, **chemical consistency**, **thermodynamic plausibility**, and **semantic alignment**.

Geometric critic. We first detect non-physical steric clashes using a covalent-radius-based minimum-distance predicate. Let $\mathcal{P}_{\mathrm{check}}(\mathcal{S})$ denote the set of atom pairs to be checked; for adsorption or molecular-complex tasks, bonded intramolecular pairs are excluded to avoid penalizing valid chemical bonds. We define

$$I_{\mathrm{geom}}(\mathcal{S}) = \prod_{(i,j) \in \mathcal{P}_{\mathrm{check}}(\mathcal{S})} \mathbb{H}\left(d_{ij}^{\mathrm{MIC}} - \gamma_t(r_i + r_j)\right), \tag{3}$$

where d_{ij}^{MIC} is the interatomic distance under the minimum-image convention when periodicity applies, r_i and r_j are covalent radii, γ_t is a task-dependent overlap tolerance, and $\mathbb{H}$ is the Heaviside function. When this predicate fails, the offending atom pairs are recorded and later verbalized as distance-correction feedback.

Chemical critic. Beyond steric feasibility, the structure should remain chemically consistent. We use oxidation-state compatibility and Bond Valence Sum (BVS)-based local valence deviation to produce

a normalized chemical-consistency score $V_{\mathrm{chem}}(\mathcal{S})$. This signal identifies chemically implausible local environments, such as inconsistent valence assignments, unreasonable coordination, or charge-imbalanced compositions. Because BVS is most informative for ionic or partially ionic systems, it is used as a soft warning signal rather than a universal chemical-validity certificate. The full BVS formulation is provided in Appendix E.

MLIP-based thermodynamic critic. Geometric and chemical checks do not capture all physically implausible cases, such as highly strained lattices, unstable adsorption placements, or poorly separated interfaces. We therefore use CHGNet [23], a pretrained machine-learning interatomic potential (MLIP), as a lightweight energy critic. Let $E_{\mathrm{norm}}(\mathcal{S}) = \mathcal{U}_\theta(\mathcal{S})/N$ denote the MLIP-predicted energy per atom. Since absolute MLIP energies are composition-dependent, we compare this value against a task- or composition-conditioned reference:

$$\Delta E_{\mathrm{MLIP}}(\mathcal{S}, \tau) = E_{\mathrm{norm}}(\mathcal{S}) - E_{\mathrm{ref}}(\tau). \tag{4}$$

We then map this relative energy to a bounded thermodynamic plausibility score:

$$V_{\mathrm{thermo}}(\mathcal{S}) = \exp(-\lambda_E \max(0,\ \Delta E_{\mathrm{MLIP}}(\mathcal{S}, \tau) - \epsilon_E))\,, \tag{5}$$

where $E_{\mathrm{ref}}(\tau)$ is obtained from a retrieved scaffold, same-composition reference, or valid initial template when available, ϵ_E is the allowed energy margin, and λ_E controls the penalty strength. Low-scoring structures are not discarded as final failures; instead, they trigger thermodynamic warnings for subsequent refinement.

Semantic alignment critic. Physical plausibility alone does not ensure that the generated structure satisfies the user's intent. We therefore parse the prompt τ into task-specific constraints, such as defect concentration, adsorption site, Miller index, vacuum thickness, CSL relation, lattice mismatch, or interface orientation. These constraints are converted into a normalized alignment score $V_{\mathrm{align}}(\mathcal{S}, \tau)$, which measures whether the current structure satisfies the requested task. The detailed constraint-to-score formulation is given in Appendix E.

Critic aggregation. The four critic outputs are aggregated into an internal diagnostic state:

$$\mathbf{z}(\mathcal{S}, \tau) = \big(I_{\mathrm{geom}}(\mathcal{S}), V_{\mathrm{chem}}(\mathcal{S}), V_{\mathrm{thermo}}(\mathcal{S}), V_{\mathrm{align}}(\mathcal{S}, \tau)\big). \tag{6}$$

This state summarizes the geometric, chemical, thermodynamic, and semantic status of the candidate structure. Rather than passing raw numerical scores directly to the LLM, MaterialsPilot uses a deterministic feedback parser to translate diagnostic violations into actionable repair instructions. The scores determine whether a violation should be triggered and how severe it is, while component-level evidence, including offending atom pairs, valence-deviation sites, MLIP energy warnings, and unmet prompt constraints, specifies the repair content. Detailed parser rules and score-to-feedback mappings are provided in Appendix F.

3.4 Iterative Execution-Feedback Optimization

We model structure generation as an iterative code optimization process. At each round, MaterialsPilot generates code, executes it, diagnoses execution or physical violations, and converts these signals into feedback for the next round. This loop allows the agent to transform runtime errors and physically invalid structures into actionable repair instructions.

Policy execution. At iteration t, the agent generates a Python program conditioned on the task specification, the current active knowledge context, and the previous interaction history:

$$c_t \sim \pi_\theta(c \mid \tau, \mathcal{K}_t, H_{t-1})\,, \tag{7}$$

where $\mathcal{K}_t$ contains retrieved task-relevant knowledge and $H_{t-1} = \{(c_i, f_i)\}_{i=0}^{t-1}$ stores previous code–feedback pairs. Executing c_t returns either a candidate structure $\mathcal{S}_t$ with construction metadata or an execution trace e_t containing syntax, API, or construction-level failures.

Feedback transformation. If execution succeeds, the physics-aware critic in Sec. 3.3 computes an internal diagnostic state $\mathbf{z}_t = \mathbf{z}(\mathcal{S}_t, \tau)$. If execution fails, we set $\mathbf{z}_t = \emptyset$ and rely on the execution trace. A deterministic feedback parser then converts the available evidence into textual repair feedback:

$$f_t = \Psi(e_t, \mathbf{z}_t, \tau)\,, \tag{8}$$

where Ψ maps runtime errors and critic-derived violations into localized instructions, such as correcting an API call, increasing interface spacing, adjusting adsorbate height, or enforcing a missing

prompt constraint. The parser is rule-based to ensure reproducibility and avoid introducing an additional learned critique model.

Feedback-aware context update. When violations are detected, the feedback signal can also trigger targeted repair retrieval. Instead of reusing a static retrieval context, MaterialsPilot expands the active knowledge set with repair-specific information:

$$\mathcal{K}_{t+1} = \mathcal{K}_t \cup \mathcal{R}_{\text{fb}}(f_t, \mathcal{K}), \qquad H_t = H_{t-1} \cup \{(c_t, f_t)\}. \tag{9}$$

For example, an overlap warning may retrieve minimum-distance or interface-spacing rules, while an adsorption-site violation may retrieve surface-site templates. The loop terminates when no execution or critic violation is detected, or when the maximum number of iterations $T_{\max}$ is reached.

4 Experiment Setting

4.1 Benchmark Tasks

We curate a benchmark of 400 tasks to evaluate language-driven atomistic construction across four topological domains, with 100 tasks per domain. The tasks span an increasing spectrum of difficulty, from local structural edits to global interface assembly. Specifically, **Defect Engineering** tests symmetry-aware site manipulation under local perturbations; **Surface Adsorption** requires rigid-body placement and SE(3)-consistent alignment on crystal surfaces; **Grain Boundaries** require deriving valid rotation matrices under coincidence site lattice (CSL) theory [24]; and **Heterostructure Interfaces** require joint reasoning over lattice mismatch, interfacial strain, and vacuum constraints. Detailed dataset statistics and prompt formulations are provided in Appendix C.

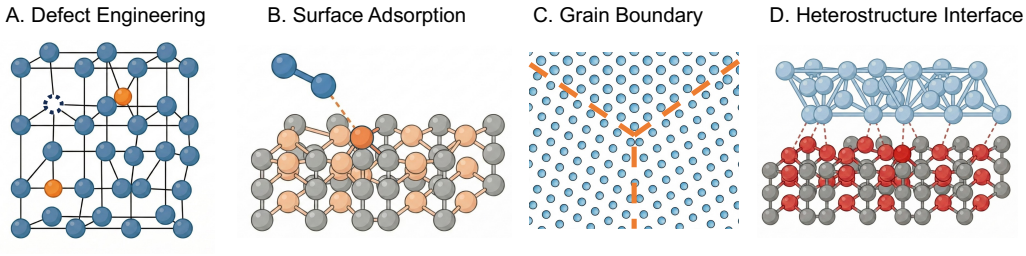

Figure 2: **Benchmark Domains Overview.** The evaluation spans four topological scales: (A) Defect Engineering (point defects); (B) Surface Adsorption (molecular placement); (C) Grain Boundary (CSL-based planar defects); and (D) Heterostructure Interface (lattice-mismatched multi-slab assembly).

4.2 Benchmark Evaluation Protocol

We evaluate all methods with the same fixed post-hoc protocol, separate from the generation-time critic in Sec. 3.3. The protocol follows a two-level logic: a generated output must first be a valid atomistic structure, and is then scored by how well it satisfies the explicit prompt requirements. All metrics are gated by a fixed structural validity indicator $I_{\text{valid}}(\mathcal{S})$, which checks structure existence, valid species, finite coordinates, and severe overlaps with task-calibrated tolerances. For each prompt τ, we parse checkable requirements into $\mathcal{C}_\tau = \{c_1, \ldots, c_{K_\tau}\}$, where c_k denotes the violation of one prompt-level constraint, such as species identity, site placement, CSL relation, interface distance, or vacuum thickness. Binary constraints use $s_k = \mathbb{I}[c_k(\mathcal{S}) \le \epsilon_k]$, while continuous constraints use normalized soft scores $s_k(\mathcal{S}, \tau) \in [0, 1]$. The per-sample constraint satisfaction is $C_{\text{sat}}(\mathcal{S}, \tau) = \frac{1}{K_\tau} \sum_{k=1}^{K_\tau} s_k(\mathcal{S}, \tau)$. We report three validity-gated metrics to comprehensively evaluate the quality of generated structures:

(1). Task Satisfaction Rate (TSR) is the strict success metric: a sample is counted only if it is valid and satisfies all prompt constraints:

$$\text{TSR} = \frac{1}{|\mathcal{D}|} \sum_{(\tau,\mathcal{S})\in\mathcal{D}} I_{\text{valid}}(\mathcal{S}) \prod_{k=1}^{K_\tau} \mathbb{I}[c_k(\mathcal{S}) \le \epsilon_k]\,. \tag{10}$$

(2). Constraint Satisfaction Rate (CSR) uses the same constraints but gives partial credit to valid near-miss structures:

$$\text{CSR} = \frac{1}{|\mathcal{D}|} \sum_{(\tau,\mathcal{S})\in\mathcal{D}} I_{\text{valid}}(\mathcal{S}) C_{\text{sat}}(\mathcal{S}, \tau). \tag{11}$$

(3). AvgScore, short for Average Score, is an auxiliary diagnostic that averages the per-sample semantic-physical quality over all valid generations:

$$\mathrm{AvgScore} = \frac{1}{|\mathcal{D}|} \sum_{(\tau,\mathcal{S})\in\mathcal{D}} I_{\mathrm{valid}}(\mathcal{S}) \left[V_{\mathrm{align}}(\mathcal{S},\tau) V_{\mathrm{chem}}(\mathcal{S}) V_{\mathrm{thermo}}(\mathcal{S})\right]^{1/3}. \tag{12}$$

Thus, invalid structures contribute zero; TSR measures full task completion, CSR measures partial constraint satisfaction, and AvgScore summarizes semantic alignment, chemical consistency, and heuristic thermodynamic plausibility. Task-specific constraint definitions are provided in Appendix D.

4.3 Baselines and Experimental Modes

To evaluate the benefit of the complete framework within code-based atomistic construction, we compare two experimental configurations: (i) **Pure Prompt (Baseline):** The LLM operates in a zero-shot setting, receiving the natural language task description and generating Python code in a single turn. This mode functions without external knowledge retrieval or execution feedback, serving as a baseline for the backbone model's intrinsic material design capability. (ii) **MaterialsPilot (Ours):** The complete agentic framework is applied on top of the backbone model. This mode activates the Hierarchical RAG (from L_1 to L_3) to inject domain-specific schemas and the Execution-Feedback Loop (max 5 iterations) to autonomously refine code based on runtime errors or physical violations.

Both main configurations generate Python code, so their comparison measures the benefit of retrieval and iterative repair within the same code-based construction interface. The Raw Feedback Loop and RAG + Raw Feedback comparisons in Appendix H.4 assess retrieval and the full repair pipeline beyond replaying unprocessed diagnostics. AtomWorld-Agent provides a task-aligned external comparison under AtomWorld's official evaluator (Sec. 5.4). Component-level conclusions rely on the corresponding controlled ablations.

We validate MaterialsPilot across seven diverse LLM architectures to confirm its generalizability and model-agnostic robustness: GPT-5.4-mini, Qwen3.6-Plus, DeepSeek-V3.2, Gemini-3.1-flash-lite, Kimi-K2.6, GLM-5.1, and MiniMax-M2.7. All models are accessed via standardized APIs with consistent generation parameters to ensure fair comparison. More details are provided in Appendix G.

5 Results

5.1 Overall Performance: A Model-Agnostic Boost

As detailed in Table 1, MaterialsPilot delivers a consistent and model-agnostic improvement across all seven evaluated LLM backbones. On average, the overall task satisfaction rate (TSR) increases from **22.0%** under direct prompting to **66.4%** with MaterialsPilot, corresponding to an absolute gain of **+44.4 percentage points**. This improvement is observed for every backbone, demonstrating that the proposed execution-feedback framework is not tied to a particular model family or scale. The strongest overall performance is achieved by Kimi-K2.6, whose TSR rises from **25.5%** to **78.5%**, while GLM-5.1 and Qwen3.6-Plus also reach competitive overall TSRs of **73.5%** and **72.5%**, respectively. Even weaker baselines benefit substantially: MiniMax-M2.7 improves from **14.0%** to **45.0%**, and Gemini-3.1-flash-lite improves from **17.0%** to **59.2%**.

The gains are also reflected in CSR, which measures validity-gated partial satisfaction of the same prompt-level constraints used by TSR. Averaged across all backbones, the overall CSR increases from **29.5%** to **77.3%**, corresponding to an absolute gain of **+47.8 percentage points**. This indicates that MaterialsPilot substantially improves partial constraint fulfillment even when strict all-or-nothing task success is not achieved. Similarly, the average overall quality score rises from **0.254** to **0.611**, corresponding to an absolute improvement of **+0.357**. These results suggest that the proposed framework does more than raise binary success rates: by iteratively checking and correcting generated structures, it helps models produce physically plausible outputs that better satisfy interatomic, geometric, and domain-specific constraints.

5.2 Ablation Studies

5.2.1 Impact of Knowledge Hierarchy

To isolate the contribution of hierarchical retrieval, we incrementally enable the L_1, L_2, and L_3 knowledge tiers while keeping the backbone model, execution-feedback loop, critic, and evaluation protocol fixed. We report CSR as the primary ablation metric because it directly measures validity-gated partial satisfaction of prompt-level constraints and is more sensitive than all-or-nothing TSR to intermediate improvements introduced by retrieved knowledge.

Table 1: **Comprehensive Performance Comparison.** Comparison of Prompt and MaterialsPilot across LLM backbones. For each model block, the three rows correspond to Prompt, MaterialsPilot, and the absolute improvement Δ, respectively. We report task satisfaction rate (TSR, %), constraint satisfaction rate (CSR, %), and average semantic-physical quality score (AvgScore, 0–1 scale) for each benchmark domain.

Model	Defect			Adsorption			Grain Boundary			Interface			Overall		
	TSR	CSR	AvgScore	TSR	CSR	AvgScore	TSR	CSR	AvgScore	TSR	CSR	AvgScore	TSR	CSR	AvgScore
GPT-5.4-mini	45.0	52.7	0.553	15.0	25.3	0.216	15.0	24.7	0.147	11.0	14.7	0.134	21.5	29.4	0.263
	79.0	**89.2**	**0.754**	**77.0**	**87.6**	**0.656**	**67.0**	**72.9**	**0.710**	**38.0**	**47.4**	**0.433**	**65.3**	**74.3**	**0.638**
	+34.0	**+36.5**	**+0.201**	**+62.0**	**+62.3**	**+0.440**	**+52.0**	**+48.2**	**+0.563**	**+27.0**	**+32.7**	**+0.299**	**+43.8**	**+44.9**	**+0.376**
Qwen3.6-Plus	45.0	49.9	0.462	26.0	35.4	0.284	34.0	41.4	0.389	21.0	21.8	0.196	31.5	37.1	0.333
	68.0	**87.2**	**0.715**	**83.0**	**87.5**	**0.599**	**69.0**	**76.2**	**0.652**	**70.0**	**73.1**	**0.615**	**72.5**	**81.0**	**0.645**
	+23.0	**+37.3**	**+0.253**	**+57.0**	**+52.1**	**+0.315**	**+35.0**	**+34.8**	**+0.263**	**+49.0**	**+51.3**	**+0.419**	**+41.0**	**+43.9**	**+0.312**
DeepSeek-V3.2	31.0	36.3	0.323	13.0	30.9	0.282	19.0	21.7	0.196	17.0	26.0	0.234	20.0	28.7	0.259
	75.0	**86.8**	**0.672**	**78.0**	**82.3**	**0.648**	**63.0**	**86.8**	**0.699**	**68.0**	**79.9**	**0.519**	**71.0**	**84.0**	**0.635**
	+44.0	**+50.5**	**+0.349**	**+65.0**	**+51.4**	**+0.366**	**+44.0**	**+65.1**	**+0.503**	**+51.0**	**+53.9**	**+0.285**	**+51.0**	**+55.2**	**+0.376**
Gemini-3.1-flash-lite	43.0	51.1	0.457	6.0	17.7	0.143	7.0	14.8	0.115	12.0	17.8	0.183	17.0	25.4	0.225
	63.0	**79.1**	**0.626**	**71.0**	**79.6**	**0.543**	**38.0**	**74.3**	**0.518**	**65.0**	**72.7**	**0.543**	**59.2**	**76.4**	**0.557**
	+20.0	**+28.0**	**+0.169**	**+65.0**	**+61.9**	**+0.400**	**+31.0**	**+59.5**	**+0.403**	**+53.0**	**+54.9**	**+0.360**	**+42.2**	**+51.1**	**+0.333**
Kimi-K2.6	39.0	39.8	0.269	28.0	45.7	0.392	15.0	28.3	0.170	20.0	29.3	0.171	25.5	35.8	0.251
	84.0	**90.2**	**0.735**	**77.0**	**91.8**	**0.718**	**80.0**	**87.5**	**0.787**	**73.0**	**78.3**	**0.506**	**78.5**	**87.0**	**0.686**
	+45.0	**+50.4**	**+0.466**	**+49.0**	**+46.1**	**+0.326**	**+65.0**	**+59.2**	**+0.617**	**+53.0**	**+49.0**	**+0.335**	**+53.0**	**+51.2**	**+0.436**
GLM-5.1	54.0	66.4	0.581	16.0	31.1	0.283	8.0	10.6	0.229	19.0	20.4	0.187	24.2	32.1	0.320
	76.0	**87.8**	**0.748**	**84.0**	**92.1**	**0.613**	**66.0**	**71.5**	**0.738**	**68.0**	**72.7**	**0.589**	**73.5**	**81.0**	**0.672**
	+22.0	**+21.4**	**+0.167**	**+68.0**	**+61.0**	**+0.330**	**+58.0**	**+60.9**	**+0.509**	**+49.0**	**+52.3**	**+0.402**	**+49.3**	**+48.9**	**+0.352**
MiniMax-M2.7	26.0	26.8	0.233	13.0	22.3	0.128	9.0	11.3	0.039	8.0	11.2	0.116	14.0	17.9	0.129
	42.0	**67.2**	**0.419**	**66.0**	**73.1**	**0.441**	**35.0**	**48.0**	**0.481**	**37.0**	**39.9**	**0.433**	**45.0**	**57.0**	**0.444**
	+16.0	**+40.4**	**+0.186**	**+53.0**	**+50.8**	**+0.313**	**+26.0**	**+36.7**	**+0.442**	**+29.0**	**+28.7**	**+0.317**	**+31.0**	**+39.1**	**+0.315**

*TSR and CSR are reported in percentages, while AvgScore is reported on a 0–1 scale. Within each model block, rows are ordered as Prompt, MaterialsPilot, and Δ from top to bottom.

Using DeepSeek-V3.2 as the controlled backbone, Table 2 shows that each knowledge tier contributes positively, but different task families exhibit different sensitivities to different forms of prior knowledge. The L_1-only setting already provides a useful foundation across all domains by enforcing compositional priors, valid species, and basic chemical constraints. Its effect is most visible in Defect Engineering, where local modifications are strongly tied to element identity, defect species, and concentration constraints. However, as the task shifts from local edits to topology-dependent construction, additional structural priors become increasingly important.

Adding L_2 produces the dominant gain for topology-sensitive tasks. Grain Boundary and Interface generation benefit most from structural schemas such as CSL transformations, interface matching rules, slab alignment, and geometry-aware construction protocols. This indicates that these tasks are not limited by chemical identity alone, but by the need to organize multiple periodic objects under strict crystallographic and interfacial constraints.

Table 2: **Ablation of Hierarchical Knowledge Tiers.** Validity-gated CSR (%) obtained by incrementally enabling L_1, L_2, and L_3 retrieval tiers using the DeepSeek-V3.2 backbone. All other modules are kept fixed.

Task	L_1	+L_2	+L_3	Δ_{2-1}	Δ_{3-2}	Δ_{3-1}
Defect	75.3	83.9	86.8	8.6	2.9	**11.5**
Adsorption	46.2	66.1	82.3	19.9	16.2	**36.1**
Grain Boundary	32.6	83.2	86.8	50.6	3.6	**54.2**
Interface	23.1	69.2	79.9	46.1	10.7	**56.8**
Avg.	44.3	75.6	84.0	31.3	8.4	**39.7**

Adding L_3 provides a smaller but complementary improvement by refining task-specific functional preferences. Its effect is especially apparent in Adsorption, where functional priors such as adsorption-site preference, adsorbate configuration, and physically reasonable placement help improve structures that are already chemically and geometrically plausible. Overall, the hierarchy improves performance across all task families, but the dominant source of improvement depends on the task: L_1 supports local chemical validity, L_2 resolves global structural feasibility, and L_3 improves functional and task-specific alignment.

5.2.2 Impact of Physics-Informed Validation

To evaluate the effect of MLIP-based thermodynamic validation, we compare MaterialsPilot with and without the MLIP feedback signal using DeepSeek-V3.2 as the backbone model. This ablation isolates the role of the MLIP critic while keeping the retrieval system, execution-feedback loop, backbone model, and post-hoc evaluation protocol fixed.

As shown in Fig. 3A, MLIP feedback produces the most consistent improvement in AvgScore, indicating that it primarily strengthens the semantic-physical quality of generated structures. In contrast, TSR and CSR exhibit smaller and task-dependent changes, suggesting that MLIP validation does not simply increase hard pass rates, but instead refines the physical plausibility of candidates that may already satisfy some prompt-level constraints. Fig. 3B illustrates this effect through a hydrogen adsorption case study. The input prompt asks the model to construct a Pt surface with a single H atom placed on the surface. Without MLIP feedback, the generated H adsorbate is positioned too far from the substrate, with a nearest adsorbate–surface distance of 5.01 Å. Such a separation is substantially larger than typical Pt–H chemisorption distances reported for atomic hydrogen on Pt, where DFT studies give Pt–H bond lengths of approximately 1.87–1.88 Å for hollow-site adsorption, while weakly interacting molecular H_2 configurations show Pt–H separations larger than 3.5 Å [25]. With MLIP-guided refinement, the nearest H–Pt distance is reduced to 1.80 Å, bringing the adsorbate into the expected chemisorption length scale and improving both CSR and AvgScore. This geometric correction is also consistent with first-principles studies showing that the preferred H adsorption geometry on Pt is sensitive to adsorption site, zero-point energy, and coverage effects [26].

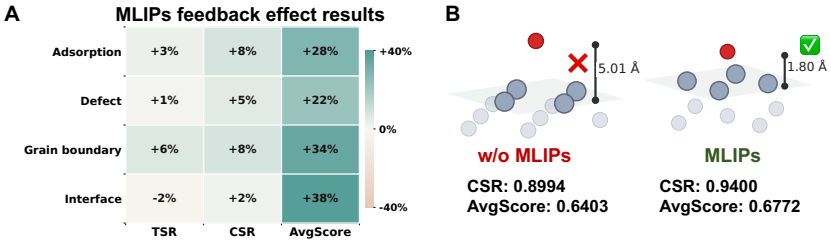

Figure 3: **Impact of MLIP-based Physics-Informed Validation.** (A) Metric changes in TSR, CSR, and AvgScore across benchmark tasks after adding MLIP feedback. (B) Hydrogen adsorption case study: Without MLIP feedback, the generated H adsorbate adopts a less realistic adsorption configuration on the Pt surface, resulting in lower CSR/AvgScore. In contrast, MLIP-guided refinement places the H atom closer to a physically plausible Pt adsorption site, better reflecting realistic hydrogen adsorption behavior.

This case study should be viewed as a qualitative illustration of MLIP-guided physical refinement rather than a standalone adsorption-structure benchmark. Because the equilibrium H–Pt distance depends on facet, adsorption site, coverage, and relaxation protocol, we use the MLIP critic only to steer an evidently under-bound geometry toward a literature-consistent chemisorption regime, while keeping the fixed post-hoc evaluation unchanged.

5.3 Qualitative Analysis on Interface Generation

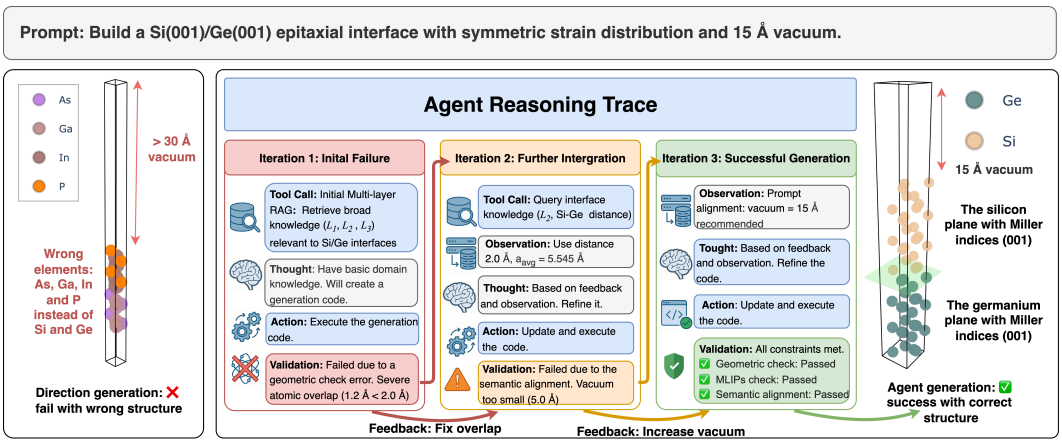

Figure 4: **Qualitative interface-generation case study.** Left: the baseline model generates a task-inconsistent III–V composition instead of the requested Si/Ge interface. Right: MaterialsPilot constructs the target through iterative repair, first correcting a severe interfacial overlap ($d_{\min} = 1.2$ Å) and then enforcing the requested 15 Å vacuum constraint.

We analyze the construction trajectory of a Si(001)/Ge(001) epitaxial interface to illustrate how MaterialsPilot handles coupled compositional, topological, and semantic constraints. This task requires enforcing the specified Si/Ge material identities, constructing a valid interface geometry, and satisfying the user-specified 15 Å vacuum.

Baseline failure: compositional hallucination. As shown in Fig. 4 (Left), the baseline model generates a task-inconsistent III–V composition containing In, Ga, As, and P, despite the explicit Si/Ge instruction. This suggests that, without retrieval grounding, the model may rely on common epitaxial-interface patterns rather than enforcing the specified material identities.

Iterative repair through execution feedback. In contrast, MaterialsPilot anchors the composition through L_1 retrieval and repairs structural errors through the execution-feedback loop (Fig. 4, Right). The repair trajectory proceeds in three stages. First, the initial generation identifies the target crystallographic setting but produces a severe interfacial overlap, with $d_{\min} = 1.2$ Å. The geometric critic detects this violation and produces localized feedback to increase the interface separation. Second, after the interface distance is adjusted to 2.0 Å, the structure becomes geometrically more feasible but defaults to a 5.0 Å vacuum, violating the requested 15 Å constraint. The semantic alignment critic flags this unmet prompt requirement and instructs the agent to explicitly enforce the vacuum thickness in the construction code. Third, guided by this feedback, the agent modifies the script to set `vacuum=15.0`, yielding a final structure that satisfies the specified material identities, interface contact, and vacuum constraint.

This case study illustrates the role of executable generation as a repair scaffold: retrieval prevents composition-level hallucination, while critic-guided feedback converts otherwise silent geometric and semantic errors into explicit, localized repair signals.

5.4 External Generalization on AtomWorld

To assess transfer beyond our four task families and custom evaluation rules, we evaluate MaterialsPilot on the independently developed AtomWorld benchmark [10], which tests natural-language-guided manipulation of atomistic structures. We use DeepSeek-V3.2 as the common backbone for Pure Prompt, AtomWorld-Agent, and MaterialsPilot.

Table 3: **AtomWorld generalization.**

Method	Success rate (%)
Pure Prompt	60.72
AtomWorld-Agent	65.38
MaterialsPilot	**78.88**

Before evaluation, we freeze the knowledge base, tool set, retrieval configuration, feedback parser, validator thresholds, and iteration budget, without adding AtomWorld-specific workflow branches. During generation, the system receives only the initial CIF structure and the manipulation instruction. Target structures are accessed exclusively by AtomWorld's official evaluator after generation and are never exposed during retrieval, code generation, validation, or repair. We report AtomWorld's official success rate rather than our TSR, CSR, or AvgScore.

As shown in Table 3, MaterialsPilot achieves a success rate of 78.88%, exceeding Pure Prompt and AtomWorld-Agent by 18.16 and 13.50 percentage points, respectively. These results support transfer of the frozen framework to externally defined structure-manipulation tasks under independent evaluation. MLIP extrapolation and open-ended materials discovery remain outside the scope of this evaluation. The complete operation-level results are provided in Appendix H.5. Performance remains dependent on knowledge coverage and evaluator reliability, especially for out-of-distribution material systems (Appendix I).

6 Conclusion

We presented MaterialsPilot, an execution-feedback framework for language-driven atomistic construction. By combining hierarchical retrieval with physics-aware iterative code refinement, MaterialsPilot reduces common geometric, chemical, and semantic failures in open-loop LLM generation. Across four task families and seven LLM backbones, it improves average TSR from 22.0% to 66.4% and CSR from 29.5% to 77.3% under a fixed post-hoc evaluation protocol. These results suggest that executable generation provides an effective path toward more constraint-satisfying and physically plausible atomistic models.

Acknowledgments and Disclosure of Funding

Bo Han was supported by the RGC Research Fellow Scheme (No. RFS2627-2S02), the RGC Young Collaborative Research Grant (No. C2005-24Y), the RGC General Research Fund (Nos. 12202026 and 12200725), the NSFC Major Research Plan (No. 92570109), the NSFC General Program (No. 62376235), the HKBU Faculty Niche Research Areas (No. RC-FNRA-IG/25-26/SCI/05), and the HKBU CSD Departmental Incentive Scheme. Tongliang Liu was partially supported by the Australian Research Council through projects DP260102466 and FT220100318. Yi Chang was supported by the Ministry of Science and Technology of China (No. 2023YFF0905400), the National Natural Science Foundation of China (No. U2341229), and the New Cornerstone Science Foundation through the XPLORER PRIZE. We extend our special thanks to Mr. Chenhan Wang from Hyper.AI for generously providing the computing resources essential to this research.

The authors have no competing interests to declare.

References

[1] Claudio Zeni, Robert Pinsler, Daniel Zügner, Andrew Fowler, Matthew Horton, Xiang Fu, Sasha Shysheya, Jonathan Crabbé, Lixin Sun, Jake Smith, et al. Mattergen: a generative model for inorganic materials design. *arXiv preprint arXiv:2312.03687*, 2023.

[2] Tian Xie, Xiang Fu, Octavian-Eugen Ganea, Regina Barzilay, and Tommi Jaakkola. Crystal diffusion variational autoencoder for periodic material generation. *arXiv preprint arXiv:2110.06197*, 2021.

[3] Richard Tran, Janice Lan, Muhammed Shuaibi, Brandon M Wood, Siddharth Goyal, Abhishek Das, Javier Heras-Domingo, Adeesh Kolluru, Ammar Rizvi, Nima Shoghi, et al. The open catalyst 2022 (oc22) dataset and challenges for oxide electrocatalysts. *ACS Catalysis*, 13(5):3066–3084, 2023.

[4] Jehad Abed, Jiheon Kim, Muhammed Shuaibi, Brook Wander, Boris Duijf, Suhas Mahesh, Hyeonseok Lee, Vahe Gharakhanyan, Sjoerd Hoogland, Erdem Irtem, et al. Open catalyst experiments 2024 (ocx24): Bridging experiments and computational models. *arXiv preprint arXiv:2411.11783*, 2024.

[5] Jean-Francois Lutz, Jean-Marie Lehn, EW Meijer, and Krzysztof Matyjaszewski. From precision polymers to complex materials and systems. *Nature Reviews Materials*, 1(5):1–14, 2016.

[6] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

[7] Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

[8] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

[9] Di Zhang, Wei Liu, Qian Tan, Jingdan Chen, Hang Yan, Yuliang Yan, Jiatong Li, Weiran Huang, Xiangyu Yue, Wanli Ouyang, et al. Chemllm: A chemical large language model. *arXiv preprint arXiv:2402.06852*, 2024.

[10] Taoyuze Lv, Alexander Chen, Fengyu Xie, Chu Wu, Jeffrey Meng, Dongzhan Zhou, Yingheng Wang, Bram Hoex, Zhicheng Zhong, and Tong Xie. Atomworld: A benchmark for evaluating spatial reasoning in large language models on crystalline materials. *arXiv preprint arXiv:2510.04704*, 2025.

[11] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

[12] Xiaoxuan Wang, Ziniu Hu, Pan Lu, Yanqiao Zhu, Jieyu Zhang, Satyen Subramaniam, Arjun R Loomba, Shichang Zhang, Yizhou Sun, and Wei Wang. Scibench: Evaluating college-level scientific problem-solving abilities of large language models. *arXiv preprint arXiv:2307.10635*, 2023.

[13] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[14] Siyu Liu, Bo Hu, Beilin Ye, Jiamin Xu, David J Srolovitz, and Tongqi Wen. Mattools: Benchmarking large language models for materials science tools. *arXiv preprint arXiv:2505.10852*, 2025.

[15] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[16] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[17] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898, 2023.

[18] Ge Lei, Ronan Docherty, and Samuel J Cooper. Materials science in the era of large language models: a perspective. *Digital Discovery*, 3(7):1257–1272, 2024.

[19] Luis M Antunes, Keith T Butler, and Ricardo Grau-Crespo. Crystal structure generation with autoregressive large language modeling. *Nature Communications*, 15(1):10570, 2024.

[20] Jingru Gan, Peichen Zhong, Yuanqi Du, Yanqiao Zhu, Chenru Duan, Haorui Wang, Daniel Schwalbe-Koda, Carla P Gomes, Kristin Persson, and Wei Wang. Large language models are innate crystal structure generators. In *AI for Accelerated Materials Design-ICLR 2025*, 2025.

[21] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[22] Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson, and Gerbrand Ceder. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013.

[23] Bowen Deng, Peichen Zhong, KyuJung Jun, Janosh Riebesell, Kevin Han, Christopher J Bartel, and Gerbrand Ceder. Chgnet as a pretrained universal neural network potential for charge-informed atomistic modelling. *Nature Machine Intelligence*, 5(9):1031–1041, 2023.

[24] S Ranganathan. On the geometry of coincidence-site lattices. *Acta Crystallographica*, 21(2):197–199, 1966.

[25] Caoming Yu, Fang Wang, Yunlei Zhang, Leihong Zhao, Botao Teng, Maohong Fan, and Xiaona Liu. H2 thermal desorption spectra on pt (111): a density functional theory and kinetic monte carlo simulation study. *Catalysts*, 8(10):450, 2018.

[26] Tran Thi Thu Hanh, Yoshinari Takimoto, and Osamu Sugino. First-principles thermodynamic description of hydrogen electroadsorption on the pt (111) surface. *Surface science*, 625:104–111, 2014.

[27] Raymond P Iczkowski and John L Margrave. Electronegativity. *Journal of the American Chemical Society*, 83(17):3547–3551, 1961.

[28] Martin Rahm, Roald Hoffmann, and NW Ashcroft. Atomic and ionic radii of elements 1–96. *Chemistry–A European Journal*, 22(41):14625–14632, 2016.

[29] Sason S Shaik and Philippe C Hiberty. *A chemist's guide to valence bond theory*. John Wiley & Sons, 2007.

[30] Calvin Ku and Patrick H-L Sit. Oxidation-state constrained density functional theory for the study of electron-transfer reactions. *Journal of Chemical Theory and Computation*, 15(9):4781–4789, 2019.

[31] George F Koster. Space groups and their representations. In *Solid state physics*, volume 5, pages 173–256. Elsevier, 1957.

[32] H Wondratschek. Splitting of wyckoff positions (orbits). *Mineralogy and Petrology*, 48(2):87–96, 1993.

[33] IJ Beyerlein, MJ Demkowicz, A Misra, and BP Uberuaga. Defect-interface interactions. *Progress in Materials Science*, 74:125–210, 2015.

[34] Vahe Tshitoyan, John Dagdelen, Leigh Weston, Alexander Dunn, Ziqin Rong, Olga Kononova, Kristin A Persson, Gerbrand Ceder, and Anubhav Jain. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571(7763):95–98, 2019.

[35] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

[36] Lowik Chanussot, Abhishek Das, Siddharth Goyal, Thibaut Lavril, Muhammed Shuaibi, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, et al. Open catalyst 2020 (oc20) dataset and community challenges. *Acs Catalysis*, 11(10):6059–6072, 2021.

[37] Danny Broberg, Bharat Medasani, Nils ER Zimmermann, Guodong Yu, Andrew Canning, Maciej Haranczyk, Mark Asta, and Geoffroy Hautier. Pycdt: A python toolkit for modeling point defects in semiconductors and insulators. *Computer Physics Communications*, 226:165–179, 2018.

[38] Jens K Nørskov, Frank Abild-Pedersen, Felix Studt, and Thomas Bligaard. Density functional theory in surface chemistry and catalysis. *Proceedings of the National Academy of Sciences*, 108(3):937–943, 2011.

[39] Christoph Freysoldt, Blazej Grabowski, Tilmann Hickel, Jörg Neugebauer, Georg Kresse, Anderson Janotti, and Chris G Van de Walle. First-principles calculations for point defects in solids. *Reviews of modern physics*, 86(1):253–305, 2014.

[40] Witold Lojkowski and Hans-Jörg Fecht. The structure of intercrystalline interfaces. *Progress in Materials Science*, 45(5-6):339–568, 2000.

[41] David L Olmsted, Stephen M Foiles, and Elizabeth A Holm. Survey of computed grain boundary properties in face-centered cubic metals: I. grain boundary energy. *Acta Materialia*, 57(13):3694–3703, 2009.

[42] Nicolas Mounet, Marco Gibertini, Philippe Schwaller, Davide Campi, Andrius Merkys, Antimo Marrazzo, Thibault Sohier, Ivano Eligio Castelli, Andrea Cepellotti, Giovanni Pizzi, et al. Two-dimensional materials from high-throughput computational exfoliation of experimentally known compounds. *Nature nanotechnology*, 13(3):246–252, 2018.

A Introduction of Hierarchical RAG Knowledge

This section provides a comprehensive breakdown of the three-tiered knowledge base ($\mathcal{K}$) utilized in MaterialsPilot. By segregating domain knowledge into semantic strata, we ensure that the retrieval mechanism targets the appropriate level of abstraction, ranging from fundamental chemical constraints to complex functional optimization.

A.1 L_1: Compositional Knowledge (Chemical Priors)

The foundational layer, L_1, acts as the system's chemical logic engine, encapsulating element-level heuristics and periodic trends that are independent of specific crystal topologies. Its primary function is to enforce local chemical validity and provide fallback inferences for out-of-distribution queries where explicit structural data is missing.

Atomic and Ionic Descriptors. At the most granular level, this layer stores immutable atomic constants. This includes Pauling electronegativity scales [27], which are critical for assessing bond polarity, and covalent radii used to determine baseline bond distances. Crucially, it incorporates a comprehensive tensor of Shannon ionic radii [28], indexed not just by element, but by oxidation state and coordination number (CN). This allows the agent to perform context-aware steric checks, for instance, distinguishing the effective size of Fe^{2+} in a tetrahedral site versus Fe^{3+} in an octahedral site.

Valence and Oxidation Rules. To ensure charge neutrality ($\sum q_i = 0$), L_1 maintains a registry of allowed oxidation states and valence bond sum (BVS) parameters [29, 30]. This data schema operates as a "soft constraint" generator during the code synthesis phase. When the agent constructs a novel stoichiometry, it queries this layer to validate whether the proposed combination of cations and anions can theoretically form a stable, charge-neutral compound, thereby pruning chemically impossible formulations before geometric construction begins.

A.2 L_2: Structural Knowledge (Topological Schemas)

Building upon valid chemical compositions, the L_2 layer defines the geometric search space. It provides the rigorous mathematical blueprints required to construct periodic lattices, transforming abstract chemical formulas into concrete crystallographic architectures.

Crystallographic Symmetry Operations. The core of this layer consists of the affine transformation matrices for all 230 space groups. Specifically, it encodes the detailed Wyckoff position multiplicities and site symmetry groups [31, 32]. This schema allows the agent to generate valid unit cells by mapping an asymmetric unit to the full lattice via symmetry operators $(\mathbf{R}, \mathbf{t})$, ensuring that the generated structures strictly adhere to international crystallographic standards.

Defect and Interface Protocols. Beyond perfect bulk crystals, L_2 encompasses schemas for topological perturbations. For defect engineering, it includes formation rules for vacancies (Schottky/Frenkel) and algorithms for supercell expansion matrices, which are necessary to isolate periodic images of defects [33]. For grain boundaries, this layer stores the Coincidence Site Lattice (CSL) theory parameters, specifically the rotation axes, angles, and Σ values required to construct mathematically valid bicrystals [24]. This enables the agent to perform complex coordinate transformations (e.g., rigid body rotations) that are mathematically precise rather than intuitively guessed.

A.3 L_3: Functional Knowledge (Property Constraints)

The highest tier, L_3, encapsulates downstream application logic. It links structural configurations to macroscopic performance metrics, serving as a functional manifold that guides the generation process toward optimal utility rather than mere validity.

Thermodynamic and Electronic Landscapes. This layer aggregates high-fidelity property data sourced from ab initio databases (e.g., Materials Project, OCP). It includes formation energies (E_{form}) and energy above the convex hull (E_{hull}) to quantify thermodynamic stability. Furthermore, it maps structures to their electronic fingerprints, such as band gap values (PBE/HSE), band edge positions, and Density of States (DOS) profiles. By retrieving from this layer, the agent can ground its generation in energy-minimizing principles, prioritizing metastable phases over high-energy artifacts.

Surface and Catalytic Descriptors. For surface-specific tasks, L_3 extends its schema to include adsorption energies (ΔG_{ads}) and surface tension metrics for specific Miller indices. This allows the system to support complex queries regarding active site selection, for example, identifying that a specific "fcc-hollow" site on a Pt(111) surface is energetically preferred for oxygen reduction. This functional grounding ensures that the generated geometries are not only spatially non-overlapping but also chemically active in the context of the target application.

A.4 Knowledge Schema Summary

Table 4 summarizes the role of each knowledge tier and the type of retrieval signal used to access it. The goal of the hierarchy is not to impose a rigid separation between chemistry, geometry, and function, but to provide the generation agent with interpretable construction priors at different levels of abstraction.

Table 4: Summary of hierarchical knowledge schema used in MaterialsPilot.

Tier	Knowledge type	Typical fields	Role in generation
L_1	Compositional priors	Elements, radii, oxidation states, BVS parameters	Enforce chemical validity
L_2	Structural schemas	Space groups, Wyckoff sites, surfaces, CSL rules, interface templates	Constrain geometry and topology
L_3	Functional constraints	Formation energy, band gap, adsorption sites, stability descriptors	Guide task-specific preferences

A.5 Knowledge-Base Construction and Curation

Source-driven construction. The knowledge base combines public materials data with source-grounded construction knowledge. Material-property and structural records are automatically imported, cleaned, and normalized from public databases. Crystallographic operations, structure-construction rules, and physical constraints are manually organized and structurally encoded using established scientific definitions, methodological literature, and materials-modeling software. The LLM is not treated as an authoritative source of material facts or numerical parameters.

The three tiers organize reusable knowledge by type rather than collecting separate answer templates for the four benchmark domains. L_1 contains elemental descriptors, oxidation-state information, bond-valence parameters, and chemical constraints. L_2 contains crystallographic operations, geometric transformations, defect operations, CSL relations, and lattice-matching rules. L_3 contains material properties and functional preferences, including information from the Materials Project and Open Catalyst Project concerning stability, adsorption, and surface-related properties.

Fixed corpus and adaptive retrieval. The LLM does not create or modify the underlying knowledge base during evaluation. Before the external AtomWorld evaluation, the knowledge base, tool set, retrieval configuration, feedback parser, validator thresholds, and iteration budget are frozen. Feedback-aware retrieval updates the active context by selecting additional existing items relevant to the current diagnostic signals; it does not modify the source corpus or introduce benchmark-specific workflow branches.

Scope and extension. The reusable units include compositional constraints, crystallographic operations, geometric transformations, and functional properties. Operations such as supercell expansion, translation, rotation, substitution, and local restructuring can be selected, parameterized, and composed for different construction tasks. Additional data sources, construction rules, or tool interfaces can be incorporated through the same five-field schema while retaining the L_1–L_3 organization.

B Details of Multi-Modal Hybrid Retrieval

This appendix provides the mathematical formulation and implementation details for the Multi-Modal Hybrid Retrieval framework introduced in Sec. 3.1. By tailoring similarity metrics to the semantic

granularity of each knowledge layer (L_1, L_2, L_3), we ensure that retrieved context is both chemically relevant and topologically isomorphic to the target task.

B.1 Query Parsing and Routing

Before retrieval, the natural language specification τ is parsed into a structured query tuple:

$$\mathrm{Parse}(\tau) = (E_\tau, T_\tau, G_\tau, P_\tau, C_\tau), \tag{13}$$

where E_τ denotes the set of chemical elements or compositions, T_τ denotes the task type, G_τ denotes geometric constraints such as Miller indices, vacuum thickness, CSL parameters, or interface orientation, P_τ denotes target physical properties, and C_τ denotes additional prompt-level constraints.

The parsed fields determine which retrieval channels are activated. For example, element sets primarily query L_1, topology-related fields such as surfaces or grain boundaries query L_2, and property or site-preference fields query L_3. If a field is absent from the prompt, the corresponding retrieval channel is still allowed but assigned lower priority during fusion. This design allows the retriever to remain robust for underspecified prompts while still exploiting detailed constraints when they are available.

B.2 Task-wise Retrieval Routing Examples

Table 5 illustrates how different benchmark tasks activate different knowledge tiers. Although all tasks may query all three tiers, the dominant retrieval signal depends on the construction objective.

Table 5: Examples of task-wise routing in hierarchical retrieval.

Task	Main query fields	Dominant tier	Retrieved knowledge
Defect engineering	Host, dopant/vacancy, concentration	L_1, L_2	Oxidation rules, supercell sizing, defect templates
Surface adsorption	Catalyst, Miller index, adsorbate, site	L_2, L_3	Slab templates, adsorption sites, height constraints
Grain boundary	Material, Σ value, axis, plane	L_2	CSL rotations, bicrystal construction rules
Heterostructure interface	Film/substrate, orientation, strain, vacuum	L_2, L_3	Lattice matching, interface spacing, strain criteria

B.3 Active Context Construction

After rank fusion, the top-ranked retrieval results are converted into a structured active context $\mathcal{K}_{\mathrm{active}}$. Each retrieved item is represented using a compact schema:

$$r_i = (\mathrm{tier}, \mathrm{source}, \mathrm{condition}, \mathrm{construction\ rule}, \mathrm{priority}), \tag{14}$$

where tier specifies whether the item comes from L_1, L_2, or L_3, $\mathrm{condition}$ describes when the rule applies, and $\mathrm{construction\ rule}$ provides the actionable information used by the code generator.

In practice, $\mathcal{K}_{\mathrm{active}}$ is organized as a short structured context rather than a flat text block. A typical entry includes: (i) chemical constraints, such as allowed oxidation states or minimum bond distances; (ii) structural construction rules, such as slab generation parameters or CSL rotations; and (iii) task-specific preferences, such as preferred adsorption sites or strain thresholds. This formatting reduces ambiguity for the LLM and encourages it to translate retrieved knowledge into executable code rather than treating it as generic background text.

B.4 Feedback-Aware Repair Retrieval

In addition to the initial retrieval from the task prompt, MaterialsPilot performs repair-specific retrieval during iterative refinement. When the feedback parser produces a textual repair signal f_t, we construct a secondary query:

$$q_t^{\mathrm{repair}} = \mathrm{Parse}(f_t, \tau), \tag{15}$$

which targets knowledge relevant to the detected failure mode. For example, a steric-overlap warning triggers retrieval of minimum-distance heuristics or interface-spacing rules; an adsorption-site violation triggers retrieval of surface-site templates; and a CSL inconsistency triggers retrieval of grain-boundary transformation rules.

The active context is then updated as:

$$\mathcal{K}_{t+1} = \mathcal{K}_t \cup \mathcal{R}_{\text{fb}}(q_t^{\text{repair}}, \mathcal{K}), \tag{16}$$

where $\mathcal{R}_{\text{fb}}$ denotes feedback-aware retrieval. This mechanism allows the agent to accumulate increasingly task-relevant knowledge across refinement rounds.

B.5 L_1: Compositional Retrieval Formulation

The compositional metric $\mathcal{S}_{\text{comp}}$ is designed to capture similarities in elemental chemistry, prioritizing exact matches while allowing for periodic trend inference. It operates on two representations:

1. Discrete Set Similarity (Jaccard Index): Let E_q and E_d be the sets of unique elements present in the query and the document, respectively. The Jaccard index $\mathcal{J}$ measures the exact overlap of elemental constituents:

$$\mathcal{J}(E_q, E_d) = \frac{|E_q \cap E_d|}{|E_q \cup E_d|} \tag{17}$$

This term effectively filters for documents containing the exact target elements (e.g., prioritizing "Si-Ge" papers for a Si/Ge interface query).

2. Continuous Embedding Similarity: To capture periodic trends (e.g., recognizing that "S" is chemically similar to "Se"), we utilize a pre-trained element embedding model, `Mat2Vec` [34]. For a composition C containing elements $\{e_1, \ldots, e_n\}$ with stoichiometric fractions $\{w_1, \ldots, w_n\}$, we compute the weighted average embedding $\mathbf{e}$:

$$\mathbf{e}_C = \sum_{i=1}^{n} w_i \cdot \text{Mat2Vec}(e_i) \tag{18}$$

The cosine similarity $\cos(\mathbf{e}_q, \mathbf{e}_d)$ then quantifies the proximity in the latent chemical space, allowing the system to retrieve chemically analogous systems (e.g., retrieving GaN templates for an AlN query) when exact matches are unavailable.

Hyperparameter: We empirically set the weighting factor $\lambda = 0.7$ to prioritize continuous chemical similarity, providing robustness against out-of-distribution queries where exact elemental matches might be sparse.

B.6 L_2: Structural Retrieval Formulation

The structural metric $\mathcal{S}_{\text{struct}}$ targets topological isomorphism. Unlike text-based retrieval which relies on keywords like "perovskite" or "fcc", we employ geometric encoders to match the underlying crystallographic symmetry.

Invariant Representations $\phi(\mathcal{S})$: We define the structural encoding $\phi(\mathcal{S})$ using a Crystal Graph Convolutional Neural Network (CGCNN) [35]. The network maps the crystal structure $\mathcal{S}$ to a fixed-size vector $\mathbf{h} \in \mathbb{R}^{64}$ that is invariant to rotation, translation, and permutation of atoms.

$$\phi(\mathcal{S}) = \text{CGCNN}_{\text{enc}}(\mathcal{S}) \tag{19}$$

This representation captures high-order topological features including coordination environments, bond angles, and lattice vector ratios.

Query Processing: Since the user query q is typically natural language, we first map it to a "prototype structure" $\mathcal{S}_q$ using a lightweight text-to-structure lookup (e.g., retrieving the canonical structure for "fcc-Cu" from the Materials Project database). The similarity is then computed in the latent structure space:

$$\mathcal{S}_{\text{struct}}(q, d) = \frac{\phi(\mathcal{S}_q) \cdot \phi(\mathcal{S}_d)}{\|\phi(\mathcal{S}_q)\| \|\phi(\mathcal{S}_d)\|} \tag{20}$$

This ensures that retrieved schemas share the same crystallographic blueprint (e.g., space group $Fm\bar{3}m$) regardless of their chemical composition.

B.7 L_3: Functional Retrieval Formulation

The functional metric $\mathcal{S}_{\text{phys}}$ operates on the physical property manifold. Its goal is to retrieve materials that match the target application's performance criteria (e.g., "semiconductor with band gap $\geq$ 2eV").

Property Vector Construction: We define a property vector $\mathbf{p} \in \mathbb{R}^k$ containing normalized scalar descriptors. For standard tasks, this includes:

$$\mathbf{p} = [\tilde{E}_{\text{form}}, \tilde{E}_{\text{gap}}, \tilde{G}_{\text{shear}}, \tilde{K}_{\text{bulk}}] \tag{21}$$

where each property is Z-score normalized using statistics from the training corpus to ensure uniform scaling.

Gaussian Kernel Similarity: To query this manifold, we parse the constraints in q (e.g., $E_{\text{gap}} \approx 1.5$) to construct a target vector $\mathbf{p}_q$. The similarity is defined using a Radial Basis Function (RBF) kernel:

$$\mathcal{S}_{\text{phys}}(q, d) = \exp\left(-\frac{\sum_i w_i (p_{q,i} - p_{d,i})^2}{2\sigma^2}\right) \tag{22}$$

where w_i allows weighting specific properties relevant to the current task (e.g., upweighting band gap for electronic queries). This effectively retrieves materials that lie in the same functional "neighborhood," serving as reference points for property optimization.

B.8 Reciprocal Rank Fusion (RRF) Justification

We employ Reciprocal Rank Fusion over linear weighted sums for two theoretical reasons:

1. **Scale Invariance:** The raw scores from the three modalities ($\mathcal{S}_{\text{comp}} \in [0, 1]$, $\mathcal{S}_{\text{struct}} \in [-1, 1]$, $\mathcal{S}_{\text{phys}} \in [0, 1]$) follow different distributions. Linear summation would require careful normalization (e.g., min-max scaling) which is sensitive to outliers. RRF operates on ranks, which are inherently scale-invariant.
2. **Robustness to Noise:** In cases where one modality fails (e.g., if the user query lacks functional constraints, $\mathcal{S}_{\text{phys}}$ becomes uninformative), RRF effectively mitigates the noise. A document must ideally be ranked highly in *at least one* informative modality to achieve a high fused score, preventing "phantom" signals from dominating the retrieval.

B.9 Example Retrieved Context

For a prompt such as "Build a Si(001)/Ge(001) epitaxial interface with symmetric strain distribution and 15Å vacuum," the retrieved active context may contain:

- L_1 **compositional constraints:** allowed species are Si and Ge; reject unrelated III–V elements; enforce minimum Si–Ge interatomic distance based on covalent radii.
- L_2 **structural schemas:** construct Si(001) and Ge(001) slabs; align surface normals along the z direction; search commensurate lateral supercells; apply symmetric strain distribution.
- L_3 **functional constraints:** prefer low-strain interface matches; enforce vacuum thickness of 15Å; avoid high-energy interface overlap configurations.

This structured context directly informs the generated Python code by specifying valid species, slab construction rules, matching constraints, and physical sanity checks.

B.10 Knowledge-Item Serialization and Delivery to the LLM

Each retrieved item uses the five-field schema $(\text{tier}, \text{source}, \text{condition}, \text{construction rule}, \text{priority})$. The source identifies the originating database, publication, standard, or method; the condition specifies its applicable elements, coordination environments, structure types, or task settings; and priority distinguishes required constraints from preferences used for candidate ranking. The following examples illustrate the field-structured representation for the three tiers.

[L_1: Chemical prior]
Source: Shannon ionic-radius table.

Condition: Fe^{3+} under sixfold coordination.
Construction rule: Use the ionic radius matched to the oxidation state and coordination environment; check local distances and reject severe atomic overlap.
Priority: Required.

[L_2: Crystallographic construction]
Source: CSL crystallographic relation.
Condition: Grain-boundary construction with a specified rotation axis and Σ value.
Construction rule: Apply the corresponding rotation and coincidence relation, generate the two oriented grains, construct the bicrystal, and remove interfacial overlap.
Priority: Required.

[L_3: Functional preference]
Source: Open Catalyst Project adsorption data.
Condition: A reported adsorbate–material/facet combination.
Construction rule: Use the source-reported adsorption site, relaxed geometry, and adsorption-energy information to rank geometrically valid adsorption configurations.
Priority: Preference.

Retrieved records are deterministically serialized into compact, field-structured natural-language context containing their sources, applicability conditions, construction rules, and priorities. They are not passed to the LLM as raw database JSON, executable functions, or complete task-specific code templates. The serialized records jointly form $\mathcal{K}_{\text{active}}$, which is combined with the user instruction and the preceding code–feedback history to condition program generation. The Si(001)/Ge(001) example above illustrates how the three knowledge tiers are combined into the active construction context.

C Details of Benchmark Datasets

C.1 Benchmark Design Philosophy: A Canonical Hierarchy

Our benchmark is not a random collection of tasks, but a structured evaluation suite designed to cover the **full hierarchy of crystallographic complexity**. It maps to the four canonical operations in computational materials science: point defect manipulation, surface functionalization, grain boundary engineering, and interface epitaxy.

Crucially, these tasks are derived from fundamental physical laws and standard literature definitions [36, 37, 38, 39, 40, 41]. By successfully navigating this spectrum ranging from local symmetry breaking to global lattice matching, a model demonstrates not merely task-specific fitting but a robust and generalized capability to reason about Euclidean geometry and periodic boundary conditions under strict physical constraints.

Scale and Diversity: To ensure statistical robustness, the benchmark comprises a total of **400 distinct sub-tasks**, evenly distributed with approximately **100 evaluation scenarios per domain**. This scale allows for a comprehensive assessment across diverse chemical compositions and structural symmetries, minimizing selection bias towards specific material families.

C.2 Benchmark Sources and Curation Pipeline

The benchmark contains 400 physical task instances, with 100 instances in each of four domains: defect engineering, surface adsorption, grain boundaries, and heterostructure interfaces. These domains span local atomic modifications, surface-bound configurations, crystallographic bicrystals, and coupled interface assembly. Material systems and parameter combinations are obtained from public candidate spaces and physical enumeration rules.

Defect engineering. Host structures are obtained from JARVIS-DFT, with vacancy records linked to JARVIS VacancyDB. Symmetry-inequivalent vacancies, interstitials, and antisites are enumerated using `pymatgen-analysis-defects`. Extrinsic substitutions are filtered using the Hautier data-driven ionic-substitution model. Compound defects are constructed according to stoichiometric or charge-conservation rules, linking the requested atomic operations to the corresponding host structures.

Surface adsorption. Material–surface–adsorbate combinations are derived from OC20, OC20-Dense, and OC20-mAds. The curation retains the source bulk identifier, Miller index, adsorbate identity, and adsorption-site information. Candidates are generated and deduplicated following the corresponding public protocols. Sampling is stratified across the in-distribution, OOD-Adsorbate, OOD-Catalyst, and OOD-Both splits to include different combinations of catalyst and adsorbate familiarity.

Grain boundaries. Candidates are collected from public Materials Project grain-boundary records. The reported material identifier, Σ value, rotation axis, rotation angle, and boundary plane define the crystallographic task parameters. Incomplete, duplicated, or excessively large structures are removed before selection. Lattice-system and boundary-type information provide the metadata used for stratification.

Heterostructure interfaces. Candidate material pairs are obtained from InterMatch and traced through the Materials Project identifiers of both components. Candidate supercells are revalidated using the Zur–McGill lattice-matching algorithm and filtered according to matching area, residual strain, and atom count. Stratification uses the dimensionality of the two components, preserving the distinction between different interface-construction settings.

Shared pipeline and instruction generation. All four domains follow a common sequence: candidate-pool construction from public databases, provenance checking, physical-validity filtering, symmetry-aware deduplication, and stratified sampling without replacement using a fixed random seed. This procedure fixes the physical instances before natural-language instructions are generated. Each selected instance is first represented as a structured task specification containing the material system, structural operation, crystallographic parameters, physical constraints, and evaluation conditions. The specification is subsequently converted into a normalized natural-language instruction. Language generation therefore changes the expression of the task rather than the underlying material selection or physical parameter combination. This ordering separates benchmark-instance construction from prompt formulation.

C.3 Detailed Task Specifications

C.3.1 Defect Engineering (Symmetry & Periodicity)

Scientific Significance: Point defects (e.g., vacancies, dopants, color centers) serve as the fundamental "qubits" and catalytic sites in modern materials [39, 37]. Precision defect engineering is non-trivial; it requires manipulating specific atomic sites to tailor electronic band structures for semiconductors and solid-state batteries.

Computational Challenge: This task tests the agent's ability to handle **Spatial Indexing and Thermodynamic Isolation**.

- *Symmetry-Aware Site Selection:* The agent must distinguish between crystallographically inequivalent Wyckoff positions. For instance, in a complex perovskite, removing an oxygen atom from an apical site versus an equatorial site results in entirely distinct physical ground states.
- *Thermodynamic Isolation (Supercell Sizing):* Unlike isolated molecular generation, solid-state defects exist in a periodic field. The agent must rigorously calculate the optimal supercell expansion matrix (e.g., determining whether a $2 \times 2 \times 2$ or $3 \times 3 \times 1$ expansion is needed) to prevent spurious self-interaction between periodic images, ensuring the simulation reflects a dilute defect limit.

Dataset Specifications: The dataset comprises 100 distinct scenarios including:

- *Vacancy Formation:* Targeted removal of atoms in high-symmetry oxides (e.g., V_{O} in TiO_2) and metals.
- *Heterovalent Doping:* Complex substitution tasks (e.g., Nitrogen-Vacancy (NV) centers in Diamond, Mg-doped GaN) that require simultaneous logic for charge compensation and local geometry relaxation.

C.3.2 Surface Adsorption

Scientific Significance: Modeling the interface between solid surfaces and gas-phase molecules is the cornerstone of heterogeneous catalysis (e.g., Hydrogen Evolution Reaction) [38]. The validity of binding energy predictions depends entirely on the precise geometric configuration of the adsorbate-slab system.

Computational Challenge: This task evaluates **SE(3) Rigid-Body Transformations on 2D Manifolds**.

- *Abstract-to-Geometric Mapping:* The agent must translate high-level chemical descriptors (e.g., "fcc-hollow", "bridge", "atop") into precise continuous coordinates. This requires parsing the surface topology to identify valid adsorption sites without strictly predefined grids.
- *Molecular Frame Alignment:* Placing a polyatomic molecule (e.g., CO, H_2O) involves a complex 6-DOF optimization. The agent must compute specific Euler angles to align molecular axes with the surface normal while strictly avoiding steric clashes (Pauli repulsion) with surface atoms, a common failure mode for non-physics-aware models.

Dataset Specifications: We focus on transition metal surfaces (Pt, Au, Cu, Ni) cleaved along low-index Miller indices [(111), (100), (110)]. The tasks require placing adsorbates ranging from simple atoms to complex rigid bodies, testing the agent's spatial reasoning in avoiding forbidden steric zones.

C.3.3 Grain Boundaries

Scientific Significance: Polycrystalline materials are dominated by grain boundaries (GBs), which govern critical macroscopic properties like fracture toughness and ionic conductivity [41]. Constructing GB models is historically error-prone due to the mathematical rigorousness required.

Computational Challenge: This is a rigorous **Matrix Derivation Task** based on Coincidence Site Lattice (CSL) theory.

- *CSL Matrix Construction:* Generation here is a mathematical transformation, not just placement. The agent must derive the exact rotation matrix $\mathbf{R}$ and rotation axis corresponding to a specific Sigma (Σ) value (reciprocal density of coinciding sites).
- *Boundary Condition Enforcement:* Merging two tilted grains requires finding a common periodic superlattice. The agent must compute integer transformation matrices to ensure that lattice points at the interface perfectly coincide, satisfying Periodic Boundary Conditions (PBC) without introducing unphysical gaps or overlaps.

Dataset Specifications: The subset includes 100 Symmetric Tilt Grain Boundaries (STGBs) in FCC and BCC systems. We explicitly select high-index boundaries (e.g., $\Sigma 5[001]$ tilt, $\Sigma 9$ boundaries) that require non-intuitive geometric transformations, serving as a stress test for the agent's mathematical reasoning capabilities.

C.3.4 Heterostructure Interfaces

Scientific Significance: Next-generation devices (e.g., 2D transistors, van der Waals heterojunctions) rely on the integration of dissimilar materials [40]. The quality of these interfaces is defined by the "lattice matching" problem—a critical bottleneck in experimental synthesis [42].

Computational Challenge: This represents a complex **Combinatorial Multi-Objective Optimization** problem.

- *Commensurate Supercell Search:* Two different materials almost never have matching lattice constants. The agent must perform a search in the integer space to find a supercell transformation matrix $\mathbf{M} = [[n_1, n_2], [n_3, n_4]]$ for both slabs.
- *Conflicting Constraints:* The goal is to minimize lattice mismatch strain (target $\epsilon < 5\%$) while simultaneously keeping the total number of atoms computationally feasible ($N < 200$). The agent must navigate this trade-off, rejecting mathematically valid but physically computationally prohibitive solutions (e.g., excessively large supercells).

Dataset Specifications: The tasks involve mating structurally distinct materials, including 2D/2D interfaces (e.g., Graphene/h-BN, MoS_2/WS_2) and challenging 2D/3D integration (e.g., Graphene on Ni(111)). This tests the agent's ability to resolve global geometric conflicts that cannot be solved by local bonding rules alone.

C.4 Prompt Formulation

All natural language prompts are synthesized using a systematic schema that typically organizes instructions into two primary components as shown in Fig. 2: (i) Task Specification: Identifies the material system and the core geometric objective. For instance, in Grain Boundary tasks, this segment defines the specific CSL type (e.g., $\Sigma 5$) and the rotation axis, requiring the agent to leverage domain knowledge to derive the corresponding transformation matrices. (ii) Geometric Constraints: Imposes explicit numerical boundaries to guarantee simulation validity. This includes defining the supercell size to ensure defect isolation (Defect Engineering), mandating vacuum layer thickness for surface decoupling (Adsorption/Slabs), and specifying strain tolerances (Heterostructures). This separation ensures the model is explicitly conditioned on all necessary physical parameters.

Defect Engineering

```
Create charge-neutral Schottky defects in NaCl with 5% concentration. Use
a 2×2×2 supercell.
```

Surface Adsorption

```
Place O atom on Pt(111) surface at top site. Create 2×2 surface slab with
4 layers and 15Å vacuum.
```

Grain Boundary

```
Create a Σ5(210) symmetric tilt grain boundary in Cu. Use rotation axis
[001] with angle 36.87°.
```

Heterostructure Interface

```
Build a Si(001)/Ge(001) epitaxial interface with symmetric strain
distribution and 15Å vacuum.
```

Figure 5: **Representative Benchmark Prompt Examples.** We display sample natural language specifications for the four evaluation domains: Defect Engineering, Surface Adsorption, Grain Boundaries, and Heterostructure Interfaces. These prompts serve as the input for the agent to autonomously translate abstract constraints (e.g., `charge-neutral`, `symmetric tilt`) into precise atomistic structures.

D Task-Specific TSR and CSR Definitions

We use a fixed post-hoc evaluation protocol that is separate from the generation-time critic. The protocol first checks structural validity and then evaluates prompt satisfaction with task-specific constraints. This separation is important because the four benchmark families encode different construction goals: a defect task asks for a localized modification to a host crystal, an adsorption task asks for a surface-bound adsorbate, a grain-boundary task asks for a bicrystal, and an interface task asks for two matched material layers in contact. The same evaluator, thresholds, and aggregation rules are applied to all methods after generation, so no model can change the scoring criteria during the feedback loop.

Structural validity. For every generated structure $\mathcal{S}$, we first apply the same validity categories with task-calibrated steric tolerances:

$$I_{\text{valid}}(\mathcal{S}) = I_{\text{exists}}(\mathcal{S}) I_{\text{atom}}(\mathcal{S}) I_{\text{coord}}(\mathcal{S}) I_{\text{steric}}(\mathcal{S}). \tag{23}$$

Here, I_{exists} checks that a structure object is produced, I_{atom} verifies valid atomic species and a non-empty atom set, I_{coord} checks finite atomic coordinates, and I_{steric} rejects severe non-physical overlaps. Periodic-cell requirements, slab geometry, vacuum thickness, CSL consistency, and interface contact are evaluated as task-level constraints rather than as part of this universal validity gate. The steric threshold is task-dependent: 1.5 Å for defect structures, 0.8 Å for adsorption structures to allow short intramolecular adsorbate bonds, and 1.0 Å for grain-boundary and interface structures.

Reported metrics. Our main tables report three post-hoc metrics, all normalized by the total number of generation attempts $|\mathcal{D}_t|$ for task family t. Invalid outputs contribute zero to all three metrics. We use structural validity only as an internal gate and do not rely on SR as a primary result. TSR, CSR, and AvgScore answer different questions: TSR measures strict task completion, CSR measures prompt-level constraint fidelity with partial credit, and AvgScore summarizes semantic, chemical, and thermodynamic quality after validity.

Task satisfaction rate. Let $\mathcal{H}_{\tau,t}$ denote the set of hard constraints for prompt τ and task type t. This is the fixed set of checkable prompt-level constraints used by the reported evaluation protocol. Each constraint $c \in \mathcal{H}_{\tau,t}$ returns a binary indicator $I_c(\mathcal{S}, \tau)$, typically by checking whether a measured violation is within a tolerance. The task satisfaction indicator is

$$I_{\text{task}}(\mathcal{S}, \tau, t) = \prod_{c \in \mathcal{H}_{\tau,t}} I_c(\mathcal{S}, \tau), \tag{24}$$

and the task satisfaction rate is

$$\text{TSR}_t = \frac{1}{|\mathcal{D}_t|} \sum_{(\tau,\mathcal{S}) \in \mathcal{D}_t} I_{\text{valid}}(\mathcal{S}) I_{\text{task}}(\mathcal{S}, \tau, t). \tag{25}$$

Thus TSR is an all-or-nothing measure over structurally valid outputs and the hard requirements of the corresponding task family. A valid structure that misses one required element, adsorption site, CSL relation, or interface-contact condition receives zero TSR for that sample even if it satisfies the remaining constraints.

Constraint satisfaction rate. TSR is intentionally strict, so we also report CSR as the partial-credit counterpart of TSR. The reported CSR uses the same prompt-level constraint set $\mathcal{H}_{\tau,t}$ as TSR; it does not introduce a separate success criterion. Each constraint $c \in \mathcal{H}_{\tau,t}$ has a partial score $s_c(\mathcal{S}, \tau) \in [0, 1]$. Binary requirements use $s_c = I_c$. Continuous requirements use pre-specified normalized scores that equal 1 inside the acceptance tolerance and decrease monotonically with the measured violation. These per-constraint scores are fixed before evaluation.

Following the main-text definition, the per-sample constraint satisfaction score is the simple average over the flattened active constraint list used by TSR:

$$C_{\text{sat}}(\mathcal{S}, \tau, t) = \frac{1}{|\mathcal{H}_{\tau,t}|} \sum_{c \in \mathcal{H}_{\tau,t}} s_c(\mathcal{S}, \tau). \tag{26}$$

The task-family CSR then averages this validity-gated quantity over all generation attempts:

$$\text{CSR}_t = \frac{1}{|\mathcal{D}_t|} \sum_{(\tau,\mathcal{S}) \in \mathcal{D}_t} I_{\text{valid}}(\mathcal{S}) C_{\text{sat}}(\mathcal{S}, \tau, t). \tag{27}$$

Invalid outputs therefore contribute zero. Constraint groups such as identity, geometry, site, CSL, and interface are used to organize Table 6 and the error analysis, but the reported CSR follows the simple average in the main text. Since TSR and CSR are computed from the same constraint set, a task-satisfying sample contributes 1 to both TSR and CSR, while a valid near miss contributes 0 to TSR but can receive a fractional CSR score. Thus reported CSR is expected to be greater than or equal to TSR under the same validity gate.

Post-validity quality score. We report AvgScore over all generation attempts, with invalid outputs assigned zero. Quality components are computed only for valid structures; invalid attempts enter the reported average as zero. For a valid output, the per-sample score is the geometric mean of semantic alignment, chemical consistency, and thermodynamic plausibility:

$$\mathrm{AvgScore}_t = \frac{1}{|\mathcal{D}_t|} \sum_{(\tau,\mathcal{S})\in\mathcal{D}_t} I_{\mathrm{valid}}(\mathcal{S}) \left[V_{\mathrm{align}}(\mathcal{S},\tau) V_{\mathrm{chem}}(\mathcal{S}) V_{\mathrm{thermo}}(\mathcal{S})\right]^{1/3}. \tag{28}$$

This makes TSR, CSR, and AvgScore comparable because all three use the same denominator: the total number of generation attempts.

Table 6: Task-specific hard constraints used for TSR and their corresponding partial satisfaction scores for CSR. TSR and CSR use the same prompt-level constraint set; CSR differs only by assigning partial credit.

Task	Hard constraints for TSR	Partial satisfaction scores for CSR
Defect	Host/material identity; required original elements; required introduced species; explicit observable element counts; charge neutrality when requested; stoichiometry preservation; minimum atom count when specified; task-specific geometry proxies for layered, dumbbell, split-vacancy, and linear dopant-center prompts.	Partial scores for identity, defect operation, chemistry, geometry, and size over the same active constraints. Continuous deviations such as concentration mismatch receive clipped soft scores.
Adsorption	Catalyst metal; adsorbate elements and adsorbate stoichiometry; a slab-like surface with at least the requested number of layers; prompt-specified Miller facet when specified; adsorbate-specific height; requested adsorption-site class (*top*, *bridge*, *fcc/hcp/hollow*).	Partial scores for material identity, adsorbate composition, slab validity, facet, adsorbate height, and adsorption-site placement over the same active constraints. Site, facet, and height deviations are scored continuously when applicable.
Grain boundary	Target material purity; bicrystal structure; minimum thickness; CSL relation for the requested Σ value, plane, and boundary type when computable; requested GB plane, rotation axis, rotation angle, and structure type.	Partial scores for material identity, bicrystal geometry, slab thickness, boundary type, and CSL consistency over the same active constraints. When exact CSL verification is infeasible, a fixed bounded proxy is used as the satisfaction score for the CSL constraint.
Interface	Film and substrate material identity; interface-type composition; sufficient vacuum; minimum slab-layer count; film-substrate contact distance in the task-specified range; requested interfacial bonding pair when specified.	Partial scores for film/substrate identity, interface composition, vacuum thickness, slab-layer count, contact distance, and interfacial bonding when present in the same active constraint set.

Worked examples. The following examples illustrate how TSR and CSR are computed from the same constraint set. The first example is pedagogical: it uses a simple prompt and explicit constraint scores to show the arithmetic.

Pedagogical interface prompt. Suppose the prompt is: *Build a Si(001)/Ge(001) interface with at least 15 Å vacuum and a film–substrate contact distance between 1.8 and 2.5 Å.* The evaluator parses four active constraints: material identity, interface composition, vacuum, and contact geometry. Consider a valid generated structure that contains separated Si and Ge slabs with sufficient vacuum, but places the two slabs too far apart, giving only partial credit for the contact-distance constraint. Its per-constraint scores might be

$$s_{\mathrm{identity}} = 1.00, \quad s_{\mathrm{composition}} = 1.00, \quad s_{\mathrm{vacuum}} = 1.00, \quad s_{\mathrm{contact}} = 0.40.$$

The hard task indicator is zero because not all active constraints pass:

$$I_{\mathrm{task}} = 1 \times 1 \times 1 \times 0 = 0,$$

so this sample contributes 0 to TSR. CSR gives partial credit over the same active constraints:

$$C_{\text{sat}} = (1.00 + 1.00 + 1.00 + 0.40)/4 = 0.85.$$

Thus a valid near miss can have $\text{TSR} = 0$ but a nonzero CSR. If the same prompt were satisfied on all four constraints, the sample would contribute 1 to both TSR and CSR.

Adsorption partial success. For an adsorption prompt, a generated slab may have the correct catalyst and adsorbate species but place the adsorbate at the wrong site class or outside the accepted height window. Such an output fails TSR if site or height is an active hard constraint, but CSR remains informative because the identity and slab constraints still receive credit. This distinguishes a chemically correct but geometrically misplaced adsorption structure from an invalid or wrong-material output.

Grain-boundary ambiguity. For a $\Sigma 5$ symmetric tilt boundary prompt, material identity and bicrystal geometry may be recoverable from the final CIF, while exact CSL reconstruction can be ambiguous for large or nonstandard cells. When deterministic CSL verification is available, it is used directly as a hard constraint. Otherwise, the reported protocol uses a fixed bounded proxy for the CSL constraint. The same proxy is binarized for TSR and used as a partial score for CSR, preserving the same-constraint relationship between the two metrics.

Implementation notes. For adsorption, the total number of atoms is not used as a reported TSR/CSR constraint unless the fixed prompt parser includes it in $\mathcal{H}_{\tau,t}$. Larger valid slabs can still satisfy the physical task when catalyst identity, adsorbate identity, slab geometry, adsorption height, site class, and any active facet constraint are correct. For interfaces, we estimate vacuum as the largest periodic gap along the surface normal and check contact using the minimum film-substrate distance. Quantities that are not included in $\mathcal{H}_{\tau,t}$, such as non-auditable construction provenance or optional representation choices, may be logged for error analysis but are not part of the reported CSR.

E Physics-Aware Critic Details

This appendix provides detailed definitions of the critic signals used by the physics-aware refinement module in Sec. 3.3. These signals are used only during iterative generation to construct textual repair feedback. They are not used as standalone final success criteria; benchmark success is evaluated separately using the fixed post-hoc protocol in Sec. 4.2.

E.1 Geometric Critic

The geometric critic detects severe steric clashes and non-physical short contacts. For periodic systems, distances are computed using the minimum-image convention when applicable. Let $\mathcal{P}_{\text{check}}(\mathcal{S})$ denote the set of atom pairs to be checked. For adsorption and molecular-complex tasks, bonded intramolecular pairs are excluded from this set to avoid penalizing valid chemical bonds.

We define:

$$I_{\text{geom}}(\mathcal{S}) = \prod_{(i,j)\in\mathcal{P}_{\text{check}}(\mathcal{S})} \mathbb{H}\left(d_{ij}^{\text{MIC}} - \gamma_t(r_i + r_j)\right), \tag{29}$$

where d_{ij}^{MIC} is the minimum-image interatomic distance, r_i and r_j are covalent radii, γ_t is a task-dependent overlap tolerance, and $\mathbb{H}$ is the Heaviside function. If $I_{\text{geom}}(\mathcal{S}) = 0$, the critic records the offending atom pairs and their distances. The feedback parser then converts this information into repair instructions such as increasing adsorbate height, enlarging interface spacing, or adjusting local atomic coordinates.

E.2 Chemical Consistency Critic

The chemical critic checks whether the generated structure is compatible with basic valence and local bonding rules. This critic is especially useful for ionic or partially ionic systems where oxidation-state assignments provide meaningful local constraints. It is used as a soft warning signal rather than as a universal chemical-validity certificate.

For atom i, the Bond Valence Sum (BVS) estimate is:

$$v_i = \sum_j \exp\left(\frac{R_{ij}^0 - d_{ij}}{b}\right), \tag{30}$$

where d_{ij} is the bond distance, R_{ij}^0 is an empirical bond-valence parameter for the atom pair, and b is a decay constant. Given a nominal oxidation state q_i, we define the average BVS deviation over atoms for which oxidation-state information is available:

$$\Delta_{\mathrm{BVS}}(\mathcal{S}) = \frac{1}{|\mathcal{I}_{\mathrm{ion}}|} \sum_{i \in \mathcal{I}_{\mathrm{ion}}} |v_i - |q_i||, \tag{31}$$

where $\mathcal{I}_{\mathrm{ion}}$ denotes the subset of atoms with assigned oxidation states and valid BVS parameters. The normalized chemical-consistency score is:

$$V_{\mathrm{chem}}(\mathcal{S}) = \exp\left(-\lambda_c \Delta_{\mathrm{BVS}}(\mathcal{S})\right). \tag{32}$$

If oxidation states are ambiguous or unavailable, the critic falls back to weaker checks such as valid species identity, stoichiometric consistency, and local coordination sanity checks. Low V_{chem} values trigger feedback such as rechecking oxidation states, correcting stoichiometry, or revising local coordination.

E.3 MLIP-based Thermodynamic Critic

The thermodynamic critic uses a pretrained machine-learning interatomic potential (MLIP), such as CHGNet, as a lightweight surrogate for identifying high-energy or highly strained candidates. This critic does not certify thermodynamic stability; it provides an inexpensive warning signal during iterative refinement.

Given the MLIP-predicted total energy $\mathcal{U}_\theta(\mathcal{S})$, we compute the energy per atom:

$$E_{\mathrm{norm}}(\mathcal{S}) = \frac{\mathcal{U}_\theta(\mathcal{S})}{N}. \tag{33}$$

Because absolute MLIP energies are composition-dependent, we compare this value against a task- or composition-conditioned reference:

$$\Delta E_{\mathrm{MLIP}}(\mathcal{S}, \tau) = E_{\mathrm{norm}}(\mathcal{S}) - E_{\mathrm{ref}}(\tau), \tag{34}$$

where $E_{\mathrm{ref}}(\tau)$ can be obtained from a retrieved scaffold, same-composition reference, or a valid initial template when available. The bounded thermodynamic plausibility score is then:

$$V_{\mathrm{thermo}}(\mathcal{S}) = \exp\left(-\lambda_E \max\left(0,\ \Delta E_{\mathrm{MLIP}}(\mathcal{S}, \tau) - \epsilon_E\right)\right), \tag{35}$$

where ϵ_E is the allowed energy margin and λ_E controls the penalty strength. For tasks where a reliable reference is unavailable, we instead use a task-conditioned admissible energy range estimated from retrieved examples or validation statistics. Low V_{thermo} values trigger warnings such as relaxing strained regions, increasing interface spacing, or avoiding unstable adsorption configurations.

E.4 Semantic Alignment Critic

The semantic critic checks whether the generated structure satisfies the explicit task requirements in the natural language specification τ. This is necessary because a structure can be physically plausible while still failing the user's intent, such as using the wrong adsorption site, missing the requested vacuum thickness, or constructing an interface with the wrong orientation.

We parse τ into task-specific constraint functions:

$$\mathcal{C}_\tau = \{c_1, \ldots, c_K\}, \tag{36}$$

where each $c_k(\mathcal{S}) \geq 0$ measures the violation of one prompt-level requirement. Examples include deviations from target defect concentration, adsorption height, Miller index, CSL relation, vacuum thickness, lattice mismatch, or interface contact distance.

Each constraint is converted into a soft satisfaction score:

$$\rho_k(\mathcal{S}, \tau) = \exp\left(-\alpha_k \max(0, c_k(\mathcal{S}) - \epsilon_k)\right), \tag{37}$$

where ϵ_k is the tolerance for constraint k and α_k controls sensitivity. The overall semantic alignment score is the geometric mean:

$$V_{\text{align}}(\mathcal{S}, \tau) = \left(\prod_{k=1}^{K} \rho_k(\mathcal{S}, \tau) \right)^{1/K}. \tag{38}$$

The geometric mean penalizes cases where one critical requirement is strongly violated, making it suitable for task-driven generation.

E.5 Critic-to-Feedback Mapping

The scalar critic outputs are not directly passed to the LLM. Instead, a deterministic feedback parser converts diagnostic violations into concise textual repair instructions. This avoids introducing an additional learned critique model and makes the refinement process reproducible.

Table 7: Examples of critic-to-feedback mapping used in the deterministic feedback parser.

Diagnostic signal	Trigger condition	Textual repair instruction
Execution failure	Runtime, API, or construction error	Localize the exception and revise the corresponding code block.
Geometric overlap	$I_{\text{geom}}(\mathcal{S}) = 0$	Increase interatomic distance; adjust adsorbate height, slab separation, or interface spacing.
Chemical inconsistency	Low $V_{\text{chem}}(\mathcal{S})$	Recheck species identity, oxidation states, stoichiometry, charge balance, or local coordination.
Thermodynamic warning	Low $V_{\text{thermo}}(\mathcal{S})$	Relax strained geometry; avoid high-energy adsorption sites or unstable interface configurations.
Semantic violation	Low $V_{\text{align}}(\mathcal{S}, \tau)$	Enforce missing prompt constraints such as adsorption site, Miller index, CSL relation, interface orientation, or vacuum thickness.

When multiple violations occur simultaneously, the parser applies a dependency-aware priority rule:

$$\text{Execution} \rightarrow \text{Geometry} \rightarrow \text{Identity/Chemistry} \rightarrow \text{Thermodynamics} \rightarrow \text{Task-specific constraints}.$$

This ordering reflects the dependencies among diagnostics. Runtime failures must be fixed before any structure can be inspected. Severe geometric clashes are handled before chemical or energetic diagnostics because overlapping atoms can make BVS or MLIP estimates unreliable. Identity and chemical-consistency errors are corrected before thermodynamic refinement. Finally, once the candidate is executable, geometrically feasible, and chemically meaningful, the parser prioritizes task-specific constraints such as adsorption site, vacuum thickness, CSL relation, or interface orientation.

F Feedback Parser and Repair Retrieval Details

This appendix describes the deterministic feedback parser used in Sec. 3.4. The parser converts execution traces and critic diagnostics into concise textual repair instructions for the next code-generation step. It is designed as a rule-based transducer rather than a learned critique model, which makes the refinement process reproducible and avoids introducing an additional trainable component.

F.1 Parser Inputs and Outputs

At iteration t, the parser receives three sources of evidence: the execution trace e_t, the critic state $\mathbf{z}_t$, and the original task specification τ. If the generated program fails to execute, e_t contains the runtime exception, failed API call, or construction-level error. If execution succeeds, the physics-aware critic produces diagnostic signals over geometric, chemical, thermodynamic, and semantic constraints.

We define the feedback parser as

$$f_t = \Psi(e_t, \mathbf{z}_t, \tau), \tag{39}$$

where f_t is a textual repair message appended to the next-generation context. Rather than passing raw numerical scores directly to the LLM, Ψ converts the available evidence into structured repair instructions.

F.2 Violation Normalization

The first stage normalizes heterogeneous evidence into a set of violation records:

$$\mathcal{D}_t = \mathcal{N}(e_t, \mathbf{z}_t, \tau) = \{d_j\}_{j=1}^{J}, \tag{40}$$

where each record is represented as

$$d_j = (m_j,\ s_j,\ \ell_j,\ a_j)\,. \tag{41}$$

Here m_j denotes the violation type, s_j denotes its severity, ℓ_j denotes localized evidence, and a_j denotes the repair target. For example, a geometric violation may include the offending atom pair and interatomic distance as ℓ_j, while a semantic violation may include the missing prompt constraint, such as an incorrect adsorption site or insufficient vacuum thickness.

F.3 Dependency-Aware Prioritization

When multiple violations occur simultaneously, the parser does not verbalize all diagnostics with equal priority. Instead, it applies a dependency-aware ordering:

$$\text{Execution} \rightarrow \text{Geometry} \rightarrow \text{Identity/Chemistry} \rightarrow \text{Thermodynamics} \rightarrow \text{Task-specific constraints}.$$

This ordering reflects diagnostic dependencies. Runtime failures must be resolved before any structure can be inspected. Severe geometric clashes are handled before chemical or energetic diagnostics because overlapping atoms can make BVS or MLIP estimates unreliable. Identity and chemical-consistency errors, such as wrong elements, invalid stoichiometry, or inconsistent oxidation states, are corrected before thermodynamic refinement. Finally, once the candidate is executable, geometrically feasible, and chemically meaningful, the parser prioritizes task-specific constraints such as adsorption site, vacuum thickness, CSL relation, or interface orientation.

Table 8: Dependency-aware priority rules used by the deterministic feedback parser.

Priority level	Trigger	Feedback action
Execution	Syntax, API, or construction failure	Localize the exception and request code-level repair.
Geometry	Severe overlap or invalid interatomic distance	Adjust coordinates, adsorbate height, slab separation, or interface spacing.
Identity/Chemistry	Wrong species, invalid stoichiometry, charge imbalance, or inconsistent valence	Correct composition, oxidation states, or local coordination.
Thermodynamics	Low MLIP plausibility or high-energy warning	Relax strained regions or avoid unstable adsorption/interface configurations.
Task-specific constraints	Violated prompt requirements such as site, vacuum, CSL, or orientation	Enforce the missing task-level constraint in the generated code.

F.4 Feedback Rendering

After prioritization, the parser renders the selected violation records into textual feedback:

$$f_t = \mathrm{Render}\left(\mathrm{TopK}_{\prec}(\mathcal{D}_t)\right), \tag{42}$$

where $\prec$ denotes the dependency-aware priority order and TopK selects the most actionable violations. In our implementation, the feedback message contains three fields:

$$f_t = [\texttt{error_summary},\ \texttt{localized_evidence},\ \texttt{repair_instruction}]\,. \tag{43}$$

The `error_summary` identifies the dominant failure mode, `localized_evidence` reports the relevant atom pair, site, interface, or constraint when available, and `repair_instruction` states the required correction in natural language.

For example, a steric-overlap violation may be rendered as: "The adsorbate is too close to the surface; increase the adsorbate height and ensure all adsorbate–surface distances exceed the minimum-distance threshold." A semantic violation may be rendered as: "The generated slab does not satisfy the requested 15Å vacuum; modify the construction code to enforce the specified vacuum thickness."

F.5 Repair-Specific Retrieval

The feedback message can also trigger targeted retrieval of repair-specific knowledge. Given textual feedback f_t, we construct a repair query:

$$q_t^{\text{repair}} = \text{Parse}_{\text{repair}}(f_t, \tau), \tag{44}$$

and retrieve additional context:

$$\mathcal{K}_{t+1} = \mathcal{K}_t \cup \mathcal{R}_{\text{fb}}\left(q_t^{\text{repair}}, \mathcal{K}\right). \tag{45}$$

This makes the retrieval context adaptive rather than static. For instance, an overlap warning retrieves minimum-distance or interface-spacing rules; an adsorption-site violation retrieves surface-site templates; a CSL inconsistency retrieves grain-boundary transformation rules; and a vacuum violation retrieves slab-construction constraints.

F.6 Termination Criterion

The loop terminates when no execution failure or critic violation is detected, or when the maximum number of iterations $T_{\max}$ is reached. Equivalently, if the parser returns an empty feedback message,

$$f_t = \emptyset, \tag{46}$$

the current candidate is accepted as the generation output and passed to the fixed post-hoc benchmark evaluation.

G Experiment Details

Compute resources. All local execution, structure validation, and MLIP-based critic evaluations were performed on workstations equipped with an AMD Ryzen 9 9950X CPU with 16 cores and 32 threads, 128 GB DDR5 memory, and an NVIDIA GeForce RTX 5090 GPU with 32 GB memory. Each machine used a 4 TB PCIe Gen4 SSD for active experiment storage, with additional HDD storage for generated structures, logs, and intermediate outputs.

Table 9: LLM backbones used in our evaluation and their access sources.

Model	Access source	URL
GPT-5.4-mini	OpenAI API documentation	`https://platform.openai.com/docs/models`
Qwen3.6-Plus	Qwen model documentation	`https://qwen.ai/blog?id=qwen3.6`
DeepSeek-V3.2	DeepSeek API documentation	`https://api-docs.deepseek.com/news/news251201`
Gemini-3.1-flash-lite	Google DeepMind model documentation	`https://deepmind.google/models/gemini/`
Kimi-K2.6	Kimi model documentation	`https://www.kimi.com/ai-models/kimi-k2-6`
GLM-5.1	Z.ai model documentation	`https://z.ai/blog/glm-5.1`
MiniMax-M2.7	MiniMax model documentation	`https://www.minimax.io/models/text/m27`

LLM inference settings. All LLM backbones were accessed through their official or provider-recommended API endpoints listed in Table 9. For each model, we used the standard inference configuration recommended or provided by the corresponding API provider and fixed these settings throughout all experiments to ensure consistency across runs. In particular, generation-related hyperparameters, including temperature, top-p, maximum generation length, frequency penalty,

presence penalty, sampling strategy, and random seed when supported, were explicitly specified or recorded according to the provider-supported standard configuration. The maximum generation length was fixed to a sufficiently large value to avoid premature truncation of structured outputs while keeping the same setting for comparable runs whenever supported by the API. This protocol was designed to evaluate each model under its standard API usage setting while avoiding additional confounding factors from model-specific hyperparameter optimization or time-varying provider defaults. For each query, the same task prompt, system instructions, retrieval context, and feedback protocol were used across all LLM backbones whenever supported by the corresponding API. If an API did not expose a particular option, that option was left unspecified and recorded as unavailable rather than approximated with another provider-specific parameter. All responses were generated using stateless API calls, and no model fine-tuning, continued pretraining, or local hosting of LLM weights was performed.

The main results were evaluated across all seven LLM backbones to validate the effectiveness and model-agnostic behavior of our method under different model families and API providers. For the remaining experiments and analyses, including ablation studies and auxiliary evaluations, we used DeepSeek-V3.2 as the backbone and applied the same fixed inference protocol described above.

H Additional Experiments

H.1 Aggregate Error Analysis

We conduct an aggregate error analysis over the generated structures to better understand the main failure modes of MaterialsPilot. Rather than inspecting individual examples, we assign each failed or imperfect generation to a primary error category using the post-hoc evaluation protocol. The categories are: generation or structure-construction failure, task-constraint mismatch, low physical or MLIP-based plausibility, chemical validity warning, partial constraint satisfaction, and no major error. When multiple issues occur in the same sample, we assign the primary category using the following priority: invalid construction, task mismatch, low physical plausibility, chemical warning, and then minor constraint degradation.

This analysis assigns each final output to one mutually exclusive primary error category; it does not track instance-level error transitions before and after introducing MLIP feedback. The resulting category distributions therefore cannot establish a systematic trade-off between physical plausibility and semantic task fidelity.

Overall, the dominant errors are not uniform across the pipeline. Without MLIP feedback, a large fraction of errors come from low physical or thermodynamic plausibility: 38.7% of the generated samples fall into this category under fair post-hoc MLIP evaluation. Structure-construction failures account for another 20.6%, mostly from invalid lattice operations, failed slab/interface construction, or code execution failures. In contrast, task-constraint mismatches account for a smaller portion, 8.8%, suggesting that the baseline often follows the task specification syntactically but may still generate physically weak structures.

With MLIP feedback, the distribution shifts. Low physical-plausibility errors decrease from 38.7% to 14.0%, and construction failures decrease from 20.6% to 14.0%. The fraction of samples with no major error increases from 18.0% to 26.4%. This supports the role of MLIP feedback as a physical-quality critic: it helps reject or repair structures with poor local environments, unrealistic periodic connectivity, or unfavorable thermodynamic scores.

The remaining errors are mostly task-specific constraint mismatches rather than purely physical failures. In the MLIP-feedback setting, task-constraint mismatch becomes more visible, increasing to 19.1%. This does not indicate that MLIP feedback is harmful by itself; rather, it shows that once obviously unphysical candidates are reduced, the bottleneck shifts toward exact prompt fidelity and symbolic task constraints. Typical residual errors include incorrect adsorbate placement, composition or defect-count mismatches, inaccurate CSL boundary constraints, insufficient interface slab construction, and imperfect interface spacing.

This trend is consistent with the metric behavior. MLIP feedback improves the continuous quality score by increasing physical and thermodynamic plausibility, while TSR and CSR can remain mixed because they depend on exact task satisfaction. Therefore, MLIP feedback should be viewed as complementary to deterministic task validation. In MaterialsPilot, the two components play different

roles: MLIP feedback improves physical realism, while protocol-based checks preserve hard task requirements.

The analysis suggests two practical directions for improvement. First, task-aware repair gates should prevent MLIP-guided revisions from changing hard constraints such as composition, adsorbate identity, defect type, or boundary/interface specification. Second, specialized constructors or templates for high-complexity tasks, especially interfaces and grain boundaries, may reduce construction failures that arise from brittle lattice, slab, and array operations.

H.2 Impact of Execution-Feedback Loop

Table 10: **Impact of Execution-Feedback Loop.** Performance comparison (CSR) between the open-loop configuration (w/o Loop) and the full MaterialsPilot.

Domain	w/o	Full	Δ	Iter.
Defect	43.7%	86.8%	+43.1	2.3
Adsorption	39.4%	82.3%	+42.9	3.2
Grain Boundary	51.9%	86.8%	+34.9	1.8
Interface	44.8%	79.9%	+35.1	2.7
Overall	45.0%	**84.0%**	**1.87** ×	**2.5**

To strictly isolate the contribution of the iterative refinement mechanism, we conducted an ablation study by deactivating the execution-feedback module of MaterialsPilot. We compare the full closed-loop system against this open-loop configuration in Table 10. Since both settings use the same knowledge access and prompt structure, the performance gap directly reflects the effect of execution-guided self-correction.

Correction Efficacy. The execution-feedback loop substantially improves overall CSR, increasing performance from 45.0% in the open-loop configuration to 84.0% in the full system. This corresponds to a 1.87× relative improvement, or an absolute gain of +39.0 percentage points. The improvement is consistent across all four domains, indicating that the feedback mechanism provides a general benefit rather than being limited to a single task type.

Domain-Level Improvements. The largest absolute gains are observed in Defect and Adsorption tasks, where CSR increases by +43.1 and +42.9 percentage points, respectively. Interface construction also benefits substantially, improving from 44.8% to 79.9% (+35.1), while Grain Boundary tasks improve from 51.9% to 86.8% (+34.9). These results suggest that direct generation often produces partially valid structures, but still leaves unresolved geometric, stoichiometric, or local coordination errors. By using execution feedback to identify and correct such violations, the full system consistently converts these partial solutions into valid outputs.

Convergence Efficiency. Despite allowing multiple refinement rounds, the agent converges efficiently, requiring only 2.5 iterations on average. Adsorption requires the most refinement, with an average of 3.2 iterations, consistent with the need to satisfy local binding-site and placement constraints. In contrast, Grain Boundary tasks converge in 1.8 iterations on average, suggesting that once the relevant structural construction rules are retrieved, many errors can be corrected with relatively few feedback steps. Overall, these results show that the execution-feedback loop is both effective and efficient: it provides large CSR gains while requiring only a small number of corrective iterations.

H.3 Sensitivity to Generation-Time Validator Thresholds

We conduct a one-at-a-time sensitivity analysis using DeepSeek-V3.2 on all 400 tasks across the four benchmark domains. Each effective triggering threshold is perturbed to $0.8\times$, $1.0\times$, and $1.2\times$ its default. We vary one threshold category at a time while fixing the backbone, hierarchical retrieval, feedback parser, iteration limit, MLIP configuration, and generation settings. For every setting, we rerun the complete generation-and-feedback process and score the resulting structures using the same frozen post-hoc evaluator. Its thresholds and aggregation rules are not perturbed.

Strictness of the perturbations. For the atom-overlap coefficient γ_t, strict/default/loose settings correspond to $1.2\times/1.0\times/0.8\times$ the default, because a larger coefficient imposes a larger minimum distance. For the MLIP warning margin, continuous semantic tolerance, and chemical-consistency

trigger, strict/default/loose settings correspond to $0.8\times$ /$1.0\times$ /$1.2\times$ the default. Prompt-specified admissible ranges and discrete semantic constraints remain unchanged.

Table 11: **Sensitivity to generation-time validator thresholds.** S and L denote strict and loose settings; paired entries report S / L. The default configuration is shared across all comparisons.

Threshold category	TSR (%)	CSR (%)	AvgScore
Default	71.0	84.0	0.635
Atom overlap	69.5 / 66.3	83.6 / 80.5	0.634 / 0.608
MLIP warning	68.0 / 64.5	81.4 / 79.4	0.643 / 0.602
Semantic tolerance	69.3 / 64.0	82.7 / 78.3	0.627 / 0.593
Chemical consistency	70.8 / 68.0	83.7 / 81.5	0.635 / 0.617

Results. Across the tested perturbations, the maximum absolute changes from the default are 7.0 percentage points in TSR, 5.7 percentage points in CSR, and 0.042 in AvgScore. Strict settings remain close to the default, while looser settings generally reduce performance. A plausible explanation is that some violations no longer trigger repair under looser thresholds. The main performance conclusions are preserved within the tested range, while the observed changes show that validator thresholds influence the effectiveness of iterative repair.

H.4 Controlled Comparison with Raw-Feedback Loops

We compare MaterialsPilot with simpler feedback mechanisms on the full 400-task benchmark using DeepSeek-V3.2. All iterative settings share the same tools, execution environment, validators, thresholds, maximum iteration budget, generation configuration, and frozen post-hoc evaluator.

Controlled settings. **Pure Prompt** generates construction code without retrieval or iterative feedback. **Raw Feedback Loop** uses no hierarchical retrieval; at each iteration, the LLM receives the previous code, execution logs, and raw validator outputs. **RAG + Raw Feedback** additionally uses the same L1–L3 hierarchical retrieval as MaterialsPilot, but retains raw diagnostics without normalization, dependency ordering, error prioritization, localization, or explicit repair instructions. **Full MaterialsPilot** converts diagnostics into structured error–evidence–repair feedback and uses feedback-aware repair retrieval.

Table 12: **Controlled comparison with raw-feedback loops.** Results on the 400-task benchmark using DeepSeek-V3.2 and the same frozen post-hoc evaluator.

Setting	TSR (%)	CSR (%)	AvgScore
Pure Prompt	20.0	28.7	0.259
Raw Feedback Loop	25.7	35.8	0.299
RAG + Raw Feedback	47.0	61.3	0.448
Full MaterialsPilot	**71.0**	**84.0**	**0.635**

Results and interpretation. Raw feedback alone provides modest improvements over Pure Prompt, whereas adding hierarchical retrieval yields substantially larger gains. Compared with RAG + Raw Feedback, Full MaterialsPilot improves TSR by 24.0 percentage points, CSR by 22.7 percentage points, and AvgScore by 0.187. These results support benefits beyond directly replaying execution and validator outputs. The latter comparison evaluates the combined effects of structured diagnostic parsing, localized repair guidance, and feedback-aware repair retrieval; it does not isolate the parser's contribution alone.

H.5 Operation-Level Results on AtomWorld

Table 13 reports the complete operation-level comparison on AtomWorld under the evaluation setting described in Sec. 5.4. All methods use DeepSeek-V3.2 and AtomWorld's official evaluator. The MaterialsPilot configuration remains frozen across operation types, and target structures are available only to the post-hoc evaluator.

Table 13: Operation-level success rates (%) on AtomWorld. All methods use DeepSeek-V3.2.

Operation	Pure Prompt	AtomWorld-Agent	MaterialsPilot
Change atom	100.00	100.00	**100.00**
Remove atom	84.00	100.00	**100.00**
Add atom	83.20	63.60	**84.60**
Move atom	67.20	90.00	**92.20**
Move towards atom	73.20	89.60	**93.20**
Insert between atoms	45.60	83.00	**83.60**
Swap atoms	40.80	49.60	**63.20**
Delete below atom	43.20	26.60	**48.20**
Rotate around atom	6.80	18.00	**36.20**
Supercell	63.20	33.40	**87.60**
Overall	60.72	65.38	**78.88**

I Broader Impact and Limitations

Broader Impact. MaterialsPilot may lower the barrier for constructing complex atomistic models from natural-language specifications, helping researchers more efficiently explore defects, adsorption systems, grain boundaries, and heterostructure interfaces. By generating executable construction code and exposing validation feedback, the framework can also improve transparency and reproducibility compared with direct coordinate generation. These capabilities may benefit computational materials workflows in areas such as catalysis, energy materials, and semiconductor interfaces.

At the same time, the system is intended as an assistive modeling tool rather than an autonomous substitute for expert judgment. Automatically generated structures should still be checked by domain experts and, when needed, validated through structural relaxation, higher-fidelity simulation, or experimental evidence before being used to support scientific conclusions.

Limitations. Our evaluation focuses on four representative but still bounded classes of atomistic construction tasks. While these cover important local, surface, boundary, and interface modeling settings, they do not exhaust all materials workflows, such as amorphous systems, liquids, finite-temperature dynamics, reaction pathways, or highly disordered multi-component structures. Extending MaterialsPilot to such regimes may require additional construction schemas and task-specific validators.

The framework remains dependent on the coverage of its knowledge base and the reliability of its physical evaluators. In particular, MLIP predictions may be unreliable for out-of-distribution material systems, and heterogeneous construction tasks may require task- or domain-specific MLIPs. The AtomWorld results support transfer to externally defined structure-manipulation tasks, but do not establish MLIP extrapolation or general applicability to open-ended materials discovery.

The physics-aware critic is designed to provide useful refinement signals, not definitive proof of physical correctness. Geometric checks, bond-valence heuristics, and MLIP-based energy estimates can detect many severe failures, but they cannot replace full relaxation or first-principles validation. In addition, some residual errors remain in challenging semantic-to-geometric mappings, such as distinguishing similar adsorption sites or resolving high-mismatch interfaces. Finally, the iterative feedback loop introduces extra computation compared with one-shot prompting, although in our experiments it typically converges within a small number of refinement rounds.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the abstract and introduction are supported by the proposed framework, evaluation protocol, and experimental results reported in Sections 3–5. The paper states the scope as language-driven atomistic construction across four benchmark domains and evaluates this claim across seven LLM backbones.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a limitations section in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper introduces algorithmic formulations and evaluation metrics, but it does not present formal theoretical results in the form of theorems requiring assumptions and proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the benchmark tasks, evaluation protocol, baselines, model backbones, and main metrics in main sections and appendix. Our code is publicly available at `https://github.com/QiaolinLu/MaterialsPilot`.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code is publicly available at: `MaterialsPilot`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the benchmark domains, evaluation metrics, baseline and full-system modes, maximum feedback iterations, and evaluated LLM backbones in previous sections, with additional dataset and model-access details in the appendices.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: No error bars are shown here; but, we use a coherent setting to ensure reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We include it in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The authors have reviewed the NeurIPS Code of Ethics, and the research does not involve human subjects, private personal data, or unsafe data collection. The work focuses on computational generation and evaluation of atomistic structures.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We include a discussion of broader impacts in the appendix.

    Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a high-risk pretrained generative model, scraped dataset, or human-facing system requiring special misuse safeguards. The released assets are expected to be code and benchmark tasks for computational atomistic construction.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We respect the license of assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new assets in current submission.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subject research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of the proposed framework and experimental evaluation. The paper describes how MaterialsPilot uses LLMs to synthesize executable Python construction programs and refine them through retrieval and execution-feedback, and evaluates the framework across seven LLM backbones.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.