Architectural Blueprint for a High-Throughput Computational Platform: Predicting Anisotropic Mechanical Properties of Metal-Organic Frameworks via Machine Learning and Interactive Visualization

1. Introduction and Domain Context

1.1 The Imperative for Computational Screening in Reticular Chemistry

The discovery and deployment of Metal-Organic Frameworks (MOFs) represents one of the most significant frontiers in modern materials science. These hybrid porous materials, constructed from inorganic metal nodes coordinated to organic linkers, offer unprecedented tunability. By varying the building blocks, chemists can theoretically synthesize millions of distinct structures, each with unique porosity, surface area, and chemical functionality.[1] This structural diversity positions MOFs as premier candidates for critical industrial applications, including gas storage (hydrogen, methane), carbon capture, catalysis, and drug delivery.[2]

However, the translation of these materials from the laboratory bench to industrial deployment is frequently bottlenecked by their mechanical stability. Unlike zeolites or dense oxides, MOFs are often mechanically compliant or even fragile due to their high porosity and the directional nature of the coordination bonds. Industrial processes such as pelletization, which is required to pack MOFs into adsorption columns, subject the crystals to significant mechanical stress. If the framework collapses under this pressure, the functional porosity is lost, rendering the material useless.[3] Consequently, understanding the mechanical properties—specifically the Young's modulus (E), shear modulus (G), and hardness (H)—is a prerequisite for practical application.

Traditional experimental characterization of these properties relies on nanoindentation, a technique where a hard tip is pressed into a crystal face to measure its resistance to deformation.[4] While accurate, nanoindentation is low-throughput, requiring high-quality single crystals and significant technician time. It is unfeasible to experimentally screen the thousands of MOFs reported in the Cambridge Structural Database (CSD) or the millions of hypothetical structures generated in silico. Furthermore, MOFs are inherently anisotropic; their stiffness varies dramatically depending on the direction of the applied load relative to

the crystal lattice.[6] A measurement taken along the pore channel may yield a vastly different modulus than one taken perpendicular to the metal-oxo chains.

1.2 The Machine Learning Paradigm Shift

To address the scalability limits of experimental characterization, the field has turned to computational simulation. Density Functional Theory (DFT) and Molecular Dynamics (MD) simulations can predict mechanical properties with high fidelity, but they remain computationally expensive, often requiring thousands of CPU hours per structure.[8] This computational cost precludes their use for real-time screening of vast chemical spaces.

Machine Learning (ML) offers a transformative solution. By training regression models on datasets of structures with known mechanical properties (derived from high-fidelity simulations or experiments), ML algorithms can predict the properties of a new structure in milliseconds.[7] The challenge, however, lies not just in the algorithm, but in the representation of the material. A robust ML pipeline must capture not only the chemical composition but also the topological and geometric features of the pore network. Moreover, for anisotropic properties like Young's modulus, the model must explicitly account for the direction of the applied force. The prediction is not a scalar value but a tensor, or at least a directional magnitude dependent on the indentation vector $[uvw]$ or plane (hkl).[10]

1.3 The Need for an Integrated Visualization Platform

While ML models can churn out predictions, the "black box" nature of these algorithms often hinders their adoption by experimentalists. A chemist does not merely want a number; they need to understand *why* a structure is stable or unstable and *how* the mechanical response relates to the crystal structure. This necessitates a platform that tightly couples prediction with visualization.

Existing tools often fail to bridge the gap between "biological" visualization (optimized for proteins) and "materials" visualization (optimized for crystal lattices). Bio-centric viewers like Mol* or NGL excel at rendering ribbons and cartoons but struggle with the infinite periodicity, unit cell packing, and polyhedral representations required for MOFs.[12] Conversely, desktop tools like VESTA or Mercury offer these features but lack the web-native architecture required for a cloud-based ML platform.

This report outlines the comprehensive design of a web-based platform that fills this void. It proposes a React-based frontend leveraging **CrystVis-js**—a domain-specific visualization library—coupled with a **FastAPI** backend that orchestrates structural parsing via **pymatgen** and property prediction via **XGBoost**. The system is designed to handle .cif file uploads, validate crystallographic integrity, accept user-defined indentation vectors (Miller indices), and render these vectors explicitly within the 3D crystal lattice, providing immediate visual confirmation of the experimental geometry being simulated.

2. Theoretical Framework: Crystallography and Mechanics

To architect a software system that accurately models material properties, the underlying scientific principles must be encoded into the software's logic. This section details the mathematical and physical frameworks that will drive the backend algorithms and frontend visualizations.

2.1 Crystallographic Coordinates and Miller Indices

At the heart of the platform is the manipulation of crystal structures. A crystal is defined by a unit cell—a parallelepiped described by three lattice vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and the angles between them α, β, γ. Atomic positions within this cell are typically stored in fractional coordinates (u, v, w), where an atom's position $\mathbf{r}$ is given by $\mathbf{r} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$.[12]

The user interaction focuses on defining directions for mechanical testing. In crystallography, directions are denoted by square brackets $[uvw]$ and planes by parentheses (hkl), known as Miller indices.

- **Directions $[uvw]$**: A vector $\mathbf{r}_{uvw} = u\mathbf{a} + v\mathbf{b} + w\mathbf{c}$.
- **Planes (hkl)**: A plane that intersects the crystallographic axes at $1/h, 1/k, 1/l$.

Crucially, in cubic systems, the direction $[hkl]$ is always perpendicular to the plane (hkl). However, MOFs often crystallize in lower-symmetry systems (monoclinic, triclinic) where this orthogonality does not hold.13 For a general crystal system, the relationship between the plane normal vector $\mathbf{n}$ and the Miller indices (hkl) involves the reciprocal lattice vectors $\mathbf{a}^*, \mathbf{b}^*, \mathbf{c}^*$:

$$\mathbf{n}_{hkl} = h\mathbf{a}^* + k\mathbf{b}^* + l\mathbf{c}^*$$

The software must implement the metric tensor formalism to accurately convert these crystallographic definitions into Cartesian coordinates (x, y, z) for visualization in the WebGL environment. The conversion from a fractional vector $\mathbf{u}_{frac}$ to Cartesian $\mathbf{u}_{cart}$ uses a transformation matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{bmatrix} a & b\cos\gamma & c\cos\beta \\ 0 & b\sin\gamma & c\frac{\cos\alpha - \cos\beta\cos\gamma}{\sin\gamma} \\ 0 & 0 & V / (ab\sin\gamma) \end{bmatrix}$$

This linear algebra transformation is a critical utility function that must reside in the backend (using libraries like pymatgen) to ensure that the arrows drawn on the frontend correctly align with the crystal structure regardless of the unit cell's skew.12

2.2 Nanoindentation and Mechanical Anisotropy

The platform simulates the result of a nanoindentation experiment. In such an experiment, a tip (indenter) is driven into the material, and the Load (P) vs. Displacement (h) curve is recorded. The unloading segment of this curve is analyzed using the **Oliver-Pharr method** to extract the indentation modulus (E_{r}) and hardness (H).[4]

$$S = \frac{dP}{dh} = \frac{2}{\sqrt{\pi}} E_r \sqrt{A}$$

Where S is the stiffness (slope of unloading), A is the projected contact area, and E_r is the reduced modulus.
For MOFs, the Young's modulus is highly dependent on orientation. A study on the MOF HKUST-1 showed that the modulus could vary from ~10 GPa to lower values depending on whether the indentation was performed on the (100) or (111) faces.[6] Therefore, the ML model inputs cannot be static features of the material alone; they must include the **indentation vector** relative to the crystal axes. This requires the backend to calculate the angle between the indentation vector and the primary bond axes (e.g., metal-metal axes) or to rotate the crystal's stiffness tensor (if predicted) to the direction of interest.[10]

2.3 Machine Learning Features for MOFs

To predict these properties without running a full simulation, the platform relies on ML. The success of the ML pipeline depends on the quality of the input features (descriptors) extracted from the .cif file.

Literature suggests distinct categories of features relevant to MOF mechanics:

1. **Geometric/Pore Features**:
    - **Pore Limiting Diameter (PLD)**: The diameter of the largest sphere that can pass through the pore windows.
    - **Largest Cavity Diameter (LCD)**: The diameter of the largest spherical void within the framework.
    - **Void Fraction**: The percentage of empty space. High porosity generally correlates with lower mechanical stiffness.[17]
2. **Chemical Features**:
    - **Metal Node Properties**: Electronegativity, ionic radius, and valence of the metal cation. Stronger metal-ligand bonds typically yield stiffer frameworks.
    - **Linker Connectivity**: The number of coordination sites on the organic ligand.
3. **Topological Features**:
    - The underlying network topology (e.g., *pcu*, *dia*, *fcu*) serves as a strong predictor of

mechanical stability. For instance, frameworks with high connectivity (like *fcu* topology in UiO-66) are generally more robust than those with low connectivity.[19]

The backend pipeline must integrate algorithms to calculate these descriptors automatically upon file upload. Tools like Zeo++ or pymatgen.analysis.porosity are essential components of this computational layer.[20]

3. High-Level System Architecture

The proposed system adopts a microservices-oriented architecture, although for the initial scale, a modular monolith using containers is sufficient and easier to maintain. The separation of the frontend (presentation) from the backend (logic/computation) is strictly enforced to allow for independent scaling—for example, if the ML inference becomes a bottleneck, the backend container can be replicated behind a load balancer without touching the frontend code.

3.1 Component Diagram and Data Flow

1. **Client (Browser)**:
    - **Framework**: React 18+ (using Functional Components and Hooks).
    - **Role**: Renders the UI, manages session state (uploaded file, selected vector), and executes the WebGL rendering loop.
2. **API Gateway / Backend**:
    - **Framework**: FastAPI (Python 3.10+).
    - **Role**: Exposes REST endpoints, validates input data using Pydantic models, handles file parsing, runs geometric algorithms, and invokes the ML model.
3. **Compute Worker (Optional but Recommended)**:
    - **Technology**: Celery with Redis or RabbitMQ.
    - **Role**: Offloads heavy tasks—such as calculating the Pore Limiting Diameter or running a complex feature extraction routine—from the main API thread to prevent blocking the server during concurrent user access.
4. **Model Store**:
    - **Storage**: Local filesystem or S3-compatible storage.
    - **Role**: Stores versioned .joblib or .onnx files containing the pre-trained ML models.

3.2 Deployment Strategy

The entire stack is containerized using **Docker**.

- **Frontend Container**: A multi-stage build where the React app is compiled to static assets (npm run build) and served via **Nginx**. This ensures high performance and correct handling of client-side routing.
- **Backend Container**: Runs the FastAPI application using **Uvicorn** (an ASGI server) behind **Gunicorn** to manage worker processes. This setup is standard for high-performance

Python web services.[21]

Data Privacy and Security:
Since users may upload proprietary crystal structures (e.g., novel MOFs not yet published), the architecture must enforce data isolation.

- **Ephemeral Storage**: Uploaded .cif files should be stored in temporary directories that are cleared after the session ends or processed purely in memory if size permits.
- **HTTPS**: All traffic between client and server must be encrypted.
- **Input Validation**: Strict validation of .cif files is crucial to prevent "XML External Entity" (XXE) style attacks or buffer overflows in parsers. The backend will use PyCIFRW's validating parser to ensure the file strictly adheres to the CIF syntax before passing it to the logic layer.[22]

4. Frontend Engineering: The Visualization Layer

The frontend is the user's window into the physics of the material. The requirements for this interface are non-trivial: it must render complex crystal lattices, handle user interactions in 3D space, and bridge the gap between "abstract numbers" (Miller indices) and "visual reality" (vectors in space).

4.1 Selection of the Visualization Engine

The choice of visualization library is pivotal. The research material compares three main contenders: **Mol***, **3Dmol.js**, and **CrystVis-js**.

*Mol (Molstar)** is the current industry standard for the Protein Data Bank (PDB). It is an engineering marvel, capable of rendering millions of atoms using GPU hardware instancing and ray-casted impostors (rendering spheres as flat billboards with depth shaders). It uses a declarative state tree architecture that allows for robust state management and undo/redo capabilities.[12] However, Mol* is fundamentally **bio-centric**. Its default representations (cartoons, ribbons) are irrelevant for materials science. Crucially, handling the "infinite lattice"—a core requirement for MOF visualization—requires specific, non-obvious preset configurations (e.g., preset.structure.unitCell). Furthermore, generating specific coordination polyhedra (visualizing the metal nodes of a MOF as geometric shapes) is not a native feature and requires custom mesh generation logic.[12]

3Dmol.js is a lightweight alternative that excels in ease of use, allowing embedding via simple HTML tags. However, it struggles with performance on large structures (>100k atoms) and lacks domain-specific features like thermal ellipsoids. It is a generalist tool that hits a ceiling when advanced crystallographic functionality is needed.[23]

CrystVis-js emerges as the superior choice for this specific platform. It is explicitly architected for the **materials science** domain. Unlike bio-viewers, it treats the **crystal lattice**

and **periodicity** as first-class citizens. Key differentiators include:

- **Thermal Ellipsoids (ADPs)**: It supports the visualization of Anisotropic Displacement Parameters—football-shaped spheres indicating atomic vibration. This is a standard visualization in desktop tools like Mercury but rare in web viewers.[12]
- **Polyhedral Rendering**: It includes native logic to detect and render coordination polyhedra (e.g., SiO4 tetrahedra or Zn4O clusters in MOF-5), which is essential for visualizing the porous topology of MOFs.[12]
- **Supercell Generation**: It can easily render $3 \times 3 \times 3$ supercells to show pore connectivity, a critical visual validation for identifying continuous channels.

4.2 Integrating CrystVis-js with React

Integrating a direct DOM-manipulation library like CrystVis-js into React's virtual DOM requires a careful "escape hatch" pattern using useRef and useEffect. We cannot let React control the DOM nodes managed by CrystVis-js, as this would cause rendering conflicts.

Implementation Pattern:
The CrystalViewer component will define a div container and use a useRef to maintain a persistent reference to the visualizer instance across re-renders.

JavaScript

```
import React, { useEffect, useRef } from 'react';
import CrystVis from 'crystvis-js';

const CrystalViewer = ({ cifContent, indentationVector }) => {
  const containerRef = useRef(null);
  const visualizerRef = useRef(null);

  // Initialization Effect
  useEffect(() => {
    if (containerRef.current &&!visualizerRef.current) {
      const vis = new CrystVis(containerRef.current, 800, 600);
      vis.highlightSelected = true; // Enable atom selection
      visualizerRef.current = vis;
    }
  },);

  // Data Loading Effect
  useEffect(() => {
    if (visualizerRef.current && cifContent) {
```

```
            const loaded = visualizerRef.current.loadModels(cifContent);
            visualizerRef.current.displayModel(loaded);

            // Set Materials-Specific Defaults
            // e.g., Show Unit Cell box
            // vis.displayUnitCell(true);
        }
    }, [cifContent]);

    // Vector Drawing Effect (See Section 4.3)
    useEffect(() => {
        if (visualizerRef.current && indentationVector) {
            drawIndentationVector(visualizerRef.current, indentationVector);
        }
    }, [indentationVector]);

    return <div ref={containerRef} className="viewer-container" />;
};
```

This pattern ensures that the expensive WebGL context creation happens only once, while data updates (loading a new CIF or changing the vector) are handled reactively.[25]

4.3 The "Arrow Problem": Custom Vector Visualization

A critical requirement is visualizing the indentation direction. While CrystVis-js supports ellipsoids, the research snippets indicate no explicit API for drawing arbitrary 3D arrows.[27] However, CrystVis-js is built on top of **Three.js**. This provides a powerful workaround: we can access the underlying Three.js scene object and inject native Three.js primitives.

Technical Strategy for Arrows:

1. **Accessing the Scene**: We must inspect the CrystVis instance to find the internal reference to the Three.js scene (often exposed as visualizer.scene or similar in open-source wrappers). If encapsulated, we may need to fork the library or use a "mixin" approach to expose it.[28]
2. **Using ArrowHelper**: Three.js provides a THREE.ArrowHelper class specifically for this purpose. It takes a direction vector (normalized), an origin point, a length, and a color.[30]
3. **Coordinate Transformation**: The ArrowHelper works in Cartesian world space. The user input is in Miller indices (hkl) or lattice directions $[uvw]$. The frontend must receive the **Cartesian** coordinates of the vector from the backend (which performs the metric tensor transformation) to ensure the arrow points in the physically correct direction relative to the skewed MOF lattice.[15]

Implementation Logic:

JavaScript

```
import * as THREE from 'three';

const drawIndentationVector = (visualizer, vectorData) => {
  // vectorData comes from backend: { start: [x,y,z], dir: [dx,dy,dz], length: L }

  // 1. Access internal Three.js scene (hypothetical access point based on common patterns)
  // In CrystVis, the model view might hold the scene group
  const scene = visualizer.modelView.scene |

| visualizer.scene;

  // 2. Create Three.js Vector objects
  const dir = new THREE.Vector3(...vectorData.dir).normalize();
  const origin = new THREE.Vector3(...vectorData.start);

  // 3. Create ArrowHelper
  const arrowHelper = new THREE.ArrowHelper(dir, origin, vectorData.length, 0xff0000); // Red
arrow

  // 4. Add to scene
  scene.add(arrowHelper);

  // 5. Trigger re-render
  visualizer.render();
};
```

This hybrid approach leverages CrystVis-js for the complex crystallography (atoms, bonds, polyhedra) while using raw Three.js for the custom annotation layer, satisfying the requirement for vector visualization where the base library might fall short.[31]

4.4 User Interface Components

The UI must facilitate scientific workflows.

- **File Upload**: Use react-dropzone for handling .cif files. It should enforce file type checks (.cif extension) and size limits (e.g., 50MB to prevent browser crashes).[33]

- **Indentation Input**: A specialized form component for Miller indices (h, k, l). It should use number inputs with validation to ensure integers are provided. A dynamic "Visualize Vector" button triggers the backend calculation to update the 3D arrow *before* the user commits to a full property prediction.[34]
- **Feedback**: Toast notifications (using react-toastify) to inform users of parsing errors (e.g., "Invalid CIF syntax") or successful model loading.

5. Backend Engineering: The Computational Core

The backend is responsible for the heavy lifting: parsing the complex STAR syntax of CIF files, validating the chemistry, calculating geometric descriptors, and executing the ML inference.

5.1 FastAPI Implementation and Pydantic Validation

FastAPI is the framework of choice due to its speed and developer ergonomics. It leverages Python's type hinting to automatically validate incoming data structures.

Schema Design:
We define Pydantic models to enforce strict contracts for the API inputs and outputs.35

Python

```
from pydantic import BaseModel, Field
from typing import List, Optional

class IndentationVector(BaseModel):
    h: int = Field(..., description="Miller index h")
    k: int = Field(..., description="Miller index k")
    l: int = Field(..., description="Miller index l")
    structure_id: str

class StructureResponse(BaseModel):
    formula: str
    space_group: str
    num_atoms: int
    lattice_params: dict
    # We return the processed structure ID for subsequent calls

class PredictionResult(BaseModel):
    youngs_modulus: float
    shear_modulus: Optional[float]
```

```
    units: str = "GPa"
    confidence_score: float
```

5.2 The pymatgen Pipeline

pymatgen (Python Materials Genomics) is the engine that powers the Materials Project and is the industry standard for crystallographic analysis in Python.[20] It is vastly superior to writing a custom parser for CIF files.

CIF Handling Workflow:

1. **Parsing**: The uploaded file is read into a pymatgen.core.Structure object. This step implicitly validates the file format; if pymatgen cannot parse it, the file is likely corrupt or non-compliant.
2. **Symmetry Analysis**: Users often upload "asymmetric units" (the smallest unique part of the crystal) or supercells. To ensure consistent ML predictions, the backend must standardize the structure. We use pymatgen.symmetry.analyzer.SpacegroupAnalyzer to detect the standard primitive cell.[33] This ensures that "MOF-5" yields the same features regardless of whether the user uploaded a single unit cell or a $2 \times 2 \times 2$ expansion.
3. **Feature Extraction**: The ML model relies on specific descriptors. pymatgen allows us to compute:
    - **Density**: structure.density
    - **Average Coordination Number**: By analyzing the neighbor list.
    - **Void Fraction**: Although pymatgen has some porosity tools, integration with **Zeo++** (a specialized C++ code for porosity) via Python subprocesses is often required for high-accuracy Pore Limiting Diameter (PLD) and Largest Cavity Diameter (LCD) calculations, which are critical features for MOF mechanics.[17]

5.3 Calculating the Indentation Vector

To draw the arrow on the frontend, the backend must convert the user's (hkl) input into Cartesian coordinates. This solves the "non-orthogonal axes" problem.

Using pymatgen, we can obtain the reciprocal lattice and calculate the normal vector.

Python

```
def calculate_vector_geometry(structure, h, k, l):
    # Get reciprocal lattice
    recip_lattice = structure.lattice.reciprocal_lattice
```

```
    # Calculate normal vector in Cartesian coords
    # n = h*a* + k*b* + l*c*
    normal_cartesian = recip_lattice.get_cartesian_coords([h, k, l])

    # Normalize length for visualization purposes (e.g., 5 Angstroms)
    # Return start (center of mass or specific atom) and end points
    return {
        "start": ,
        "direction": normal_cartesian,
        "length": 10.0 # arbitrary visual scale
    }
```

This API endpoint /vector-geometry allows the frontend to be "dumb" regarding crystallography—it just draws a line from A to B—while the backend handles the complex metric tensor math.[15]

5.4 Machine Learning Inference Strategy

The ML component acts as a surrogate for physical simulation.

- **Model Choice**: **XGBoost** or **Random Forest** regressors are recommended. Literature suggests they offer the best balance of accuracy and interpretability for materials properties, outperforming simple linear models and often matching deep learning on tabular data without the massive data requirements.[7]
- **Directional Inputs**: The model input vector $\mathbf{x}$ will be a concatenation of:
    1. **Structure Features**: PLD, LCD, void fraction, metal node atomic number, linker connectivity.
    2. **Directional Features**: The direction $[uvw]$ (or plane normal) converted to spherical coordinates (θ, ϕ) or directional cosines relative to the crystal axes. This explicitly encodes the anisotropy.[38]
- **Deployment**: The trained model is serialized (pickled) using joblib. Upon server startup, the model is loaded into memory once (singleton pattern) to ensure low-latency predictions (~50ms) per request.[39]

6. Data Management and Reproducibility

6.1 Data Integrity

The system must ensure that the structure being visualized is identical to the one being analyzed.

- **Session State**: When a file is uploaded, the server assigns a unique session_id or structure_id (UUID). The parsed structure is cached (using Redis or simply in-memory dictionary for small scale) against this ID.

- **Statelessness**: The REST API remains stateless. Every request to /predict must carry the structure_id to retrieve the correct crystal context.

6.2 FAIR Principles

To align with FAIR (Findable, Accessible, Interoperable, Reusable) data principles, the platform should optionally allow users to contribute their data.

- **Metadata Storage**: If the user consents, the uploaded .cif and the predicted properties can be logged to a MongoDB database. This creates a feedback loop, effectively crowd-sourcing a dataset of "structures of interest" which can be used to retrain and improve the ML model in the future.[40]

7. Deployment and Operations

7.1 Dockerization

A docker-compose.yml file defines the service orchestration:

1. frontend: Nginx container serving the React build.
2. backend: Python container running FastAPI/Uvicorn.
3. worker (optional): Python container for heavy background tasks (Zeo++ calculations).
4. redis (optional): Message broker for the worker queue.

7.2 Testing Strategy

- **Backend Tests**: Use pytest to validate the crystallographic logic. Create "golden files" (CIFs with known properties) and assert that the pymatgen parser calculates the correct lattice parameters and that the /vector-geometry endpoint returns the correct Cartesian vectors for known Miller indices.[41]
- **Integration Tests**: Test the full loop—upload CIF $\rightarrow$ receive ID $\rightarrow$ request prediction $\rightarrow$ receive JSON response.

8. Conclusion

This report defines a rigorous, full-stack architecture for a MOF property prediction platform. By explicitly addressing the domain-specific challenges—anisotropy, crystallographic symmetry, and the "materials/bio" visualization divide—it moves beyond a generic web app to a specialized scientific instrument. The selection of **CrystVis-js** ensures visual fidelity to the chemist's mental model (ellipsoids, polyhedra), while the **FastAPI + pymatgen** backend ensures mathematical correctness. The integration of **XGBoost** provides the predictive power of modern data science without the computational cost of simulation. This platform will not only accelerate the screening of porous materials but also democratize access to advanced mechanical characterization, placing the power of supercomputing-grade predictions into the hands of experimentalists via a web browser.

Table 1: Technology Stack Summary

Component	Technology	Justification
Frontend Framework	React.js	Component-based, vast ecosystem, efficient state management.
Visualization Engine	CrystVis-js	Domain-specific features (ellipsoids, supercells) critical for MOFs.
3D Primitives	Three.js (via CrystVis)	Enables custom arrow/vector drawing for indentation visualization.
Backend API	FastAPI	High performance, native Pydantic validation, async support.
Crystallography Lib	pymatgen	Industry standard, robust symmetry & structure analysis tools.
ML Model	XGBoost (via scikit-learn/joblib)	High accuracy on tabular data, handles non-linear anisotropy well.
Containerization	Docker	Ensures reproducibility and consistent deployment environment.

Table 2: Comparison of Visualization Libraries for MOFs

Feature	Mol* (Molstar)	3Dmol.js	CrystVis-js
Primary Domain	Biology (Proteins)	General / Education	**Materials Science**

Performance	Extreme (Millions of atoms)	Moderate	High (optimized for lattices)
Unit Cell / Packing	Secondary feature	Supported	**Core feature**
Thermal Ellipsoids	Rare / Difficult	No	**Native Support**
Polyhedra	Custom Implementation	Limited	**Native Support**
Custom Vectors	Possible but complex	Supported	**Via Three.js access**
Verdict	Overkill / Wrong focus	Too basic	**Optimal**

Works cited

1. From Data to Discovery: Recent Trends of Machine Learning in Metal–Organic Frameworks - PMC - PubMed Central, accessed January 14, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC11522899/
2. The transformative role of machine learning in advancing MOF membranes for gas separations | Chemical Physics Reviews | AIP Publishing, accessed January 14, 2026, https://pubs.aip.org/aip/cpr/article/6/3/031303/3360374/The-transformative-role-of-machine-learning-in
3. Machine learning predicts mechanical properties of porous materials, accessed January 14, 2026, https://www.cam.ac.uk/research/news/machine-learning-predicts-mechanical-properties-of-porous-materials
4. Probing Soft Fibrous Materials by Indentation - PMC - NIH, accessed January 14, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC9526757/
5. How do you measure Young's Modulus with an AFM? - Asylum Research, accessed January 14, 2026, https://afm.oxinst.com/outreach/how-do-you-measure-youngs-modulus-with-an-afm
6. Mechanical properties of metal-organic frameworks: An indentation study on epitaxial thin films | Applied Physics Letters | AIP Publishing, accessed January 14, 2026, https://pubs.aip.org/aip/apl/article/101/10/101910/126727/Mechanical-properties-of-metal-organic-frameworks
7. Machine learning assisted prediction of the Young's modulus of compositionally

complex alloys - PMC - NIH, accessed January 14, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC8387451/
8. Accurate Prediction of Mechanical Property of Organic Crystals Using Molecular Dynamics-Based Nanoindentation Simulations - PMC - NIH, accessed January 14, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC12532296/
9. [2507.04493] Machine Learning-Based Prediction of Metal-Organic Framework Materials: A Comparative Analysis of Multiple Models - arXiv, accessed January 14, 2026, https://arxiv.org/abs/2507.04493
10. (PDF) A Physics Constrained Machine Learning Pipeline for Young's Modulus Prediction in Multimaterial Hyperelastic Cylinders Guided by Contact Mechanics - ResearchGate, accessed January 14, 2026, https://www.researchgate.net/publication/398959913_A_Physics_Constrained_Machine_Learning_Pipeline_for_Young's_Modulus_Prediction_in_Multimaterial_Hyperelastic_Cylinders_Guided_by_Contact_Mechanics
11. Crystal Visualizer Tutorial | PDF | Crystal Structure | Computing - Scribd, accessed January 14, 2026, https://www.scribd.com/document/941372305/Crystal-Visualizer-Tutorial
12. 3D CIF File Visualization Toolkits.pdf
13. Lattice Planes and Miller Indices (all content) - Dissemination of IT for the Promotion of Materials Science (DoITPoMS), accessed January 14, 2026, https://www.doitpoms.ac.uk/tlplib/miller_indices/printall.php
14. Tutorial: Change the Crystal Orientation - Atomsk, accessed January 14, 2026, https://atomsk.univ-lille.fr/tutorial_orient.php
15. Crystallographic calculator - Semiconductor Spectroscopy and Devices, accessed January 14, 2026, https://ssd.phys.strath.ac.uk/resources/crystallography/crystallographic-direction-calculator/
16. Mechanical properties of metal-organic frameworks: An indentation study on epitaxial thin films | Request PDF - ResearchGate, accessed January 14, 2026, https://www.researchgate.net/publication/257951822_Mechanical_properties_of_metal-organic_frameworks_An_indentation_study_on_epitaxial_thin_films
17. Machine Learning for Gas Adsorption in Metal–Organic Frameworks: A Review on Predictive Descriptors | Industrial & Engineering Chemistry Research - ACS Publications, accessed January 14, 2026, https://pubs.acs.org/doi/10.1021/acs.iecr.4c03500
18. Machine learning prediction of open metal sites in metal-organic framework catalysts - ChemRxiv, accessed January 14, 2026, https://chemrxiv.org/engage/api-gateway/chemrxiv/assets/orp/resource/item/6340dd33975e94ae1a98b279/original/machine-learning-prediction-of-open-metal-sites-in-metal-organic-framework-catalysts.pdf
19. System of Agentic AI for the Discovery of Metal-Organic Frameworks - arXiv, accessed January 14, 2026, https://arxiv.org/html/2504.14110v1
20. pymatgen: Home, accessed January 14, 2026, https://pymatgen.org/
21. WO2020263358A1 - Machine learning techniques for estimating mechanical properties of materials - Google Patents, accessed January 14, 2026,

https://patents.google.com/patent/WO2020263358A1/en?q=%22Machine+Learning+Techniques+For+Estimating+Mechanical+Properties+Of+Materials%22&oq=%22Machine+Learning+Techniques+For+Estimating+Mechanical+Properties+Of+Materials%22
22. A validating CIF parser: PyCIFRW - ResearchGate, accessed January 14, 2026, https://www.researchgate.net/publication/242134843_A_validating_CIF_parser_PyCIFRW
23. 3Dmol.js - bio.tools, accessed January 14, 2026, https://bio.tools/3dmol
24. 3dmol/3Dmol.js: WebGL accelerated JavaScript molecular graphics library - GitHub, accessed January 14, 2026, https://github.com/dkoes/3Dmol.js/
25. stur86/crystvis-js: A Three.js based crystallographic visualisation tool - GitHub, accessed January 14, 2026, https://github.com/stur86/crystvis-js
26. Integrating with Other Libraries - React, accessed January 14, 2026, https://legacy.reactjs.org/docs/integrating-with-other-libraries.html
27. crystvis-js - GitHub Pages, accessed January 14, 2026, https://ccp-nc.github.io/crystvis-js/
28. How do I get direct access to the Three.js scene? · google model-viewer · Discussion #1873 - GitHub, accessed January 14, 2026, https://github.com/google/model-viewer/discussions/1873
29. Three.js, how to access items inside scene? Should I use document.getElementById()?, accessed January 14, 2026, https://stackoverflow.com/questions/16673937/three-js-how-to-access-items-inside-scene-should-i-use-document-getelementbyid
30. ArrowHelper – three.js docs, accessed January 14, 2026, https://threejs.org/docs/pages/ArrowHelper.html
31. Arrow helpers in three.js | Dustin John Pfister at github pages, accessed January 14, 2026, https://dustinpfister.github.io/2018/11/10/threejs-arrow-helper/
32. How do you make a custom arrow? - Questions - three.js forum, accessed January 14, 2026, https://discourse.threejs.org/t/how-do-you-make-a-custom-arrow/55401
33. Understanding Structures and Properties in the Materials Project, accessed January 14, 2026, https://docs.materialsproject.org/methodology/materials-methodology/understanding-structures-and-properties-in-the-materials-project
34. Free Miller Indices Calculator - Mathos AI, accessed January 14, 2026, https://www.mathgptpro.com/app/calculator/miller-indices-calculator
35. Declare Request Example Data - FastAPI, accessed January 14, 2026, https://fastapi.tiangolo.com/tutorial/schema-extra-example/
36. Form Models - FastAPI, accessed January 14, 2026, https://fastapi.tiangolo.com/tutorial/request-form-models/
37. How do I use pymatgen in python to visualize.cif files, accessed January 14, 2026, https://matsci.org/t/how-do-i-use-pymatgen-in-python-to-visualize-cif-files/53848
38. Machine learning-based prediction of elastic properties of amorphous metal alloys - arXiv, accessed January 14, 2026, https://arxiv.org/pdf/2306.08387

39. Serving ML Models with FastAPI: A Production-Ready API in Minutes - Grigor Khachatryan, accessed January 14, 2026, https://grigorkh.medium.com/serving-ml-models-with-fastapi-a-production-ready-api-in-minutes-b5f4839a33a9
40. (PDF) A database of molecular properties integrated in the Materials Project - ResearchGate, accessed January 14, 2026, https://www.researchgate.net/publication/373059473_A_database_of_molecular_properties_integrated_in_the_Materials_Project
41. jamesrhester/pycifrw - GitHub, accessed January 14, 2026, https://github.com/jamesrhester/pycifrw