Project A: PolyKye Onchain

What **PolyKye's smart contracts** might look like for an onchain workflow, using clear explanations and some Solidity code examples.

Assumptions & Requirements

- **Users** submit a **disease target**.
- **Off-chain computation** (since AI/ML screening is too big/expensive for onchain) generates:
  - the **optimal ligand** (represented as a SMILES string or similar),
  - a **synthesis pathway** (ideally stored on IPFS/Arweave),
  - any **score/metadata**.
- **Results** are recorded onchain for provenance, transparency, and user incentives.

Core Smart Contract Functions

1. Submit Target

Allow users to submit a disease target (could be a string, ID, or reference).

```
event TargetSubmitted(address indexed user, uint indexed targetId, string target);


struct TargetSubmission {
    address user;
    string target;
    uint timestamp;
    bool processed;
}


mapping(uint => TargetSubmission) public targets;
```

```
uint public targetCount;

function submitTarget(string memory target) public returns (uint targetId) {
    targetId = targetCount++;
    targets[targetId] = TargetSubmission(msg.sender, target, block.timestamp,
false);
    emit TargetSubmitted(msg.sender, targetId, target);
}
```

2. Submit Result

After off-chain processing, results are submitted back onchain, linked to the target.

```
event ResultSubmitted(
    uint indexed targetId,
    string ligandSmiles,
    string synthesisIpfsHash,
    uint score
);

struct Result {
    string ligandSmiles;
    string synthesisIpfsHash;
    uint score;
    uint timestamp;
    address submitter;
}

mapping(uint => Result) public results; // key: targetId

function submitResult(
    uint targetId,
    string memory ligandSmiles,
    string memory synthesisIpfsHash,
    uint score
) public {
    require(targetId < targetCount, "Invalid targetId");
    require(!targets[targetId].processed, "Already processed");

    results[targetId] = Result(
        ligandSmiles,
        synthesisIpfsHash,
        score,
        block.timestamp,
        msg.sender
    );
    targets[targetId].processed = true;

    emit ResultSubmitted(targetId, ligandSmiles, synthesisIpfsHash, score);
}
```

3. Get Results

Anyone can view the output (fully transparent and onchain):

```
function getResult(uint targetId) public view returns (
    string memory ligandSmiles,
    string memory synthesisIpfsHash,
    uint score,
    uint timestamp,
    address submitter
) {
    Result storage result = results[targetId];
    return (
        result.ligandSmiles,
        result.synthesisIpfsHash,
        result.score,
        result.timestamp,
        result.submitter
    );
}
```

How This Works in Practice

- **User submits a disease target** (recorded onchain).
- **Off-chain service picks this up, runs agentic workflow,** finds best ligand/synthesis, uploads pathway file to IPFS/Arweave (getting a content hash).
- **Off-chain service (or user) calls `submitResult`** with target ID, ligand SMILES, IPFS hash, score.
- **Anyone can look up the result,** verify provenance, and access the full synthesis protocol via the IPFS hash.

Possible Extensions

- **Incentives:** Add reward for best discoveries (e.g., token payout).
- **Verification:** Use cryptographic proofs or multi-sig for result attestation.
- **Reputation:** Track successful submissions by wallet address.
- **Upgradeability:** Allow for contract upgrades if the protocol evolves.

Summary Table

Smart Contract Feature	Purpose
`submitTarget()`	Record new disease targets

`submitResult()`	Store ligands & synthesis (with IPFS)
`getResult()`	Anyone can view results & provenance
Data structures (mapping, struct)	Efficient storage for lookups
Events	Easy to track submissions/results

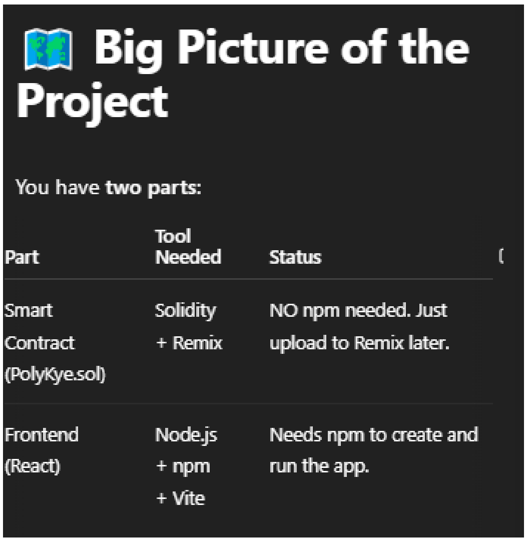
Conclusion

A **PolyKye smart contract** would:

- Store user-submitted disease targets.
- Record the optimal ligand and synthesis pathway from off-chain computation.
- Securely link to full pathway data stored on a decentralized file system (IPFS/Arweave).
- Make the workflow transparent, discoverable, and tamper-proof.

🛠️ In Practice (User Story)

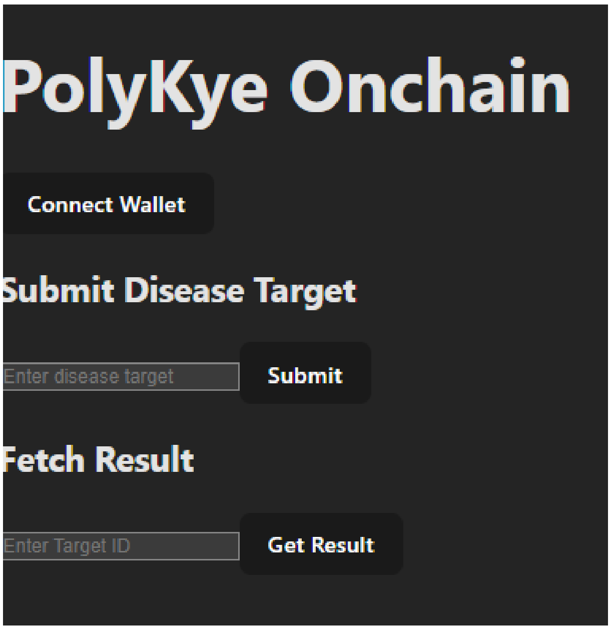
1. A researcher submits “pancreatic cancer” to the app.
2. PolyKye’s backend AI designs a molecule and synthesis route.
3. The system:
   - Uploads the synthesis to IPFS.
   - Gets the IPFS content hash.
   - Calls the smart contract’s `submitResult()` with:
     - The ligand
     - The IPFS hash
     - A score (e.g., binding affinity)
4. All data is now **permanently, transparently onchain.**
5. A future researcher or pharma company can:
   - Verify it.
   - Reproduce it.
   - Build on it.



🗺️ Big Picture of the Project

You have **two parts:**

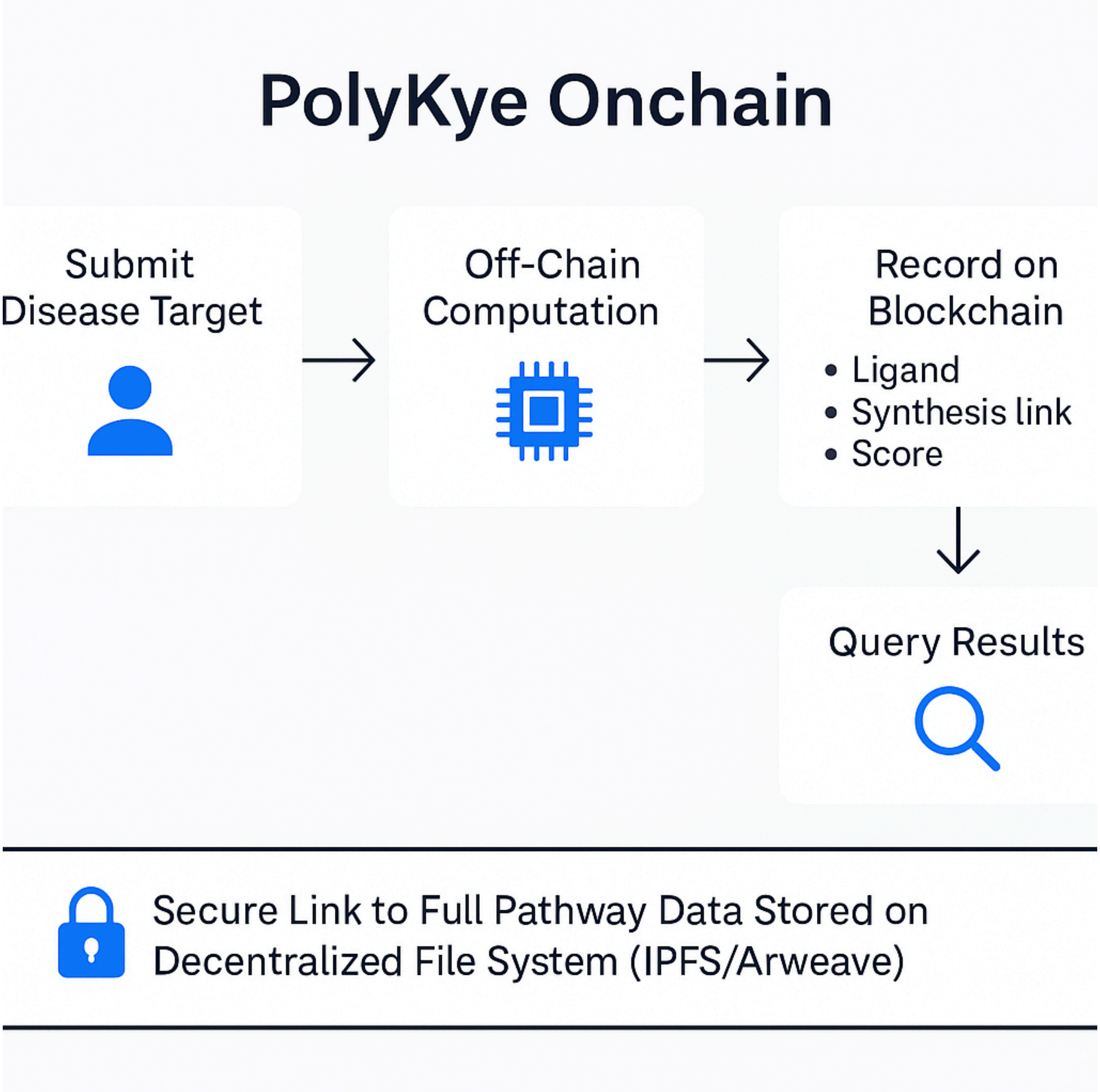
Part	Tool Needed	Status	(
Smart Contract (PolyKye.sol)	Solidity + Remix	NO npm needed. Just upload to Remix later.	
Frontend (React)	Node.js + npm + Vite	Needs npm to create and run the app.	



PolyKye Onchain

Submit Disease Target

Fetch Result



PolyKye Onchain

Submit Disease Target

Off-Chain Computation

Record on Blockchain

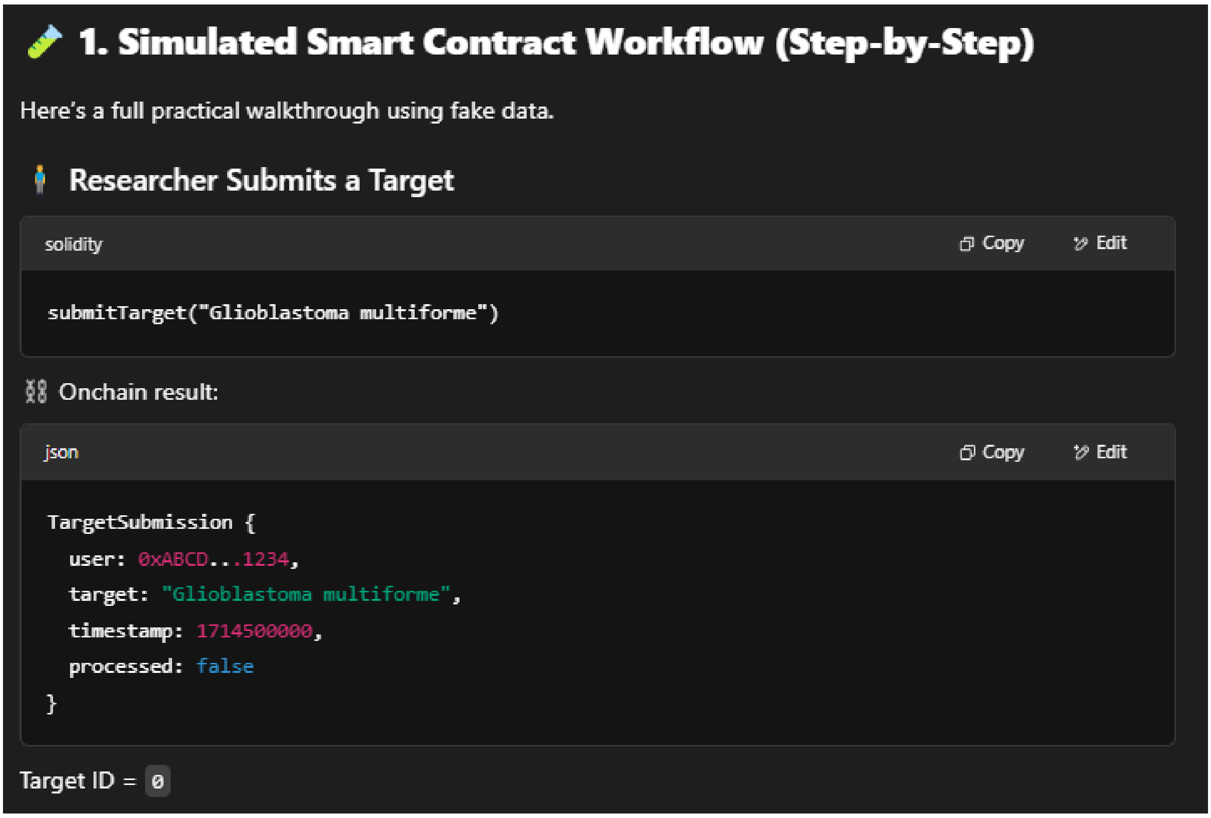
- Ligand
- Synthesis link
- Score

Query Results

Secure Link to Full Pathway Data Stored on Decentralized File System (IPFS/Arweave)

🔐 Why Use Blockchain?

- **Tamper-proof:** Results can’t be changed or faked.
- **Transparent:** Publicly verifiable research provenance.
- **Decentralized collaboration:** Enables global, bias-resistant contributions.
- **Immutable record:** Drug discovery workflows are recorded permanently.



🧪 1. Simulated Smart Contract Workflow (Step-by-Step)

Here's a full practical walkthrough using fake data.

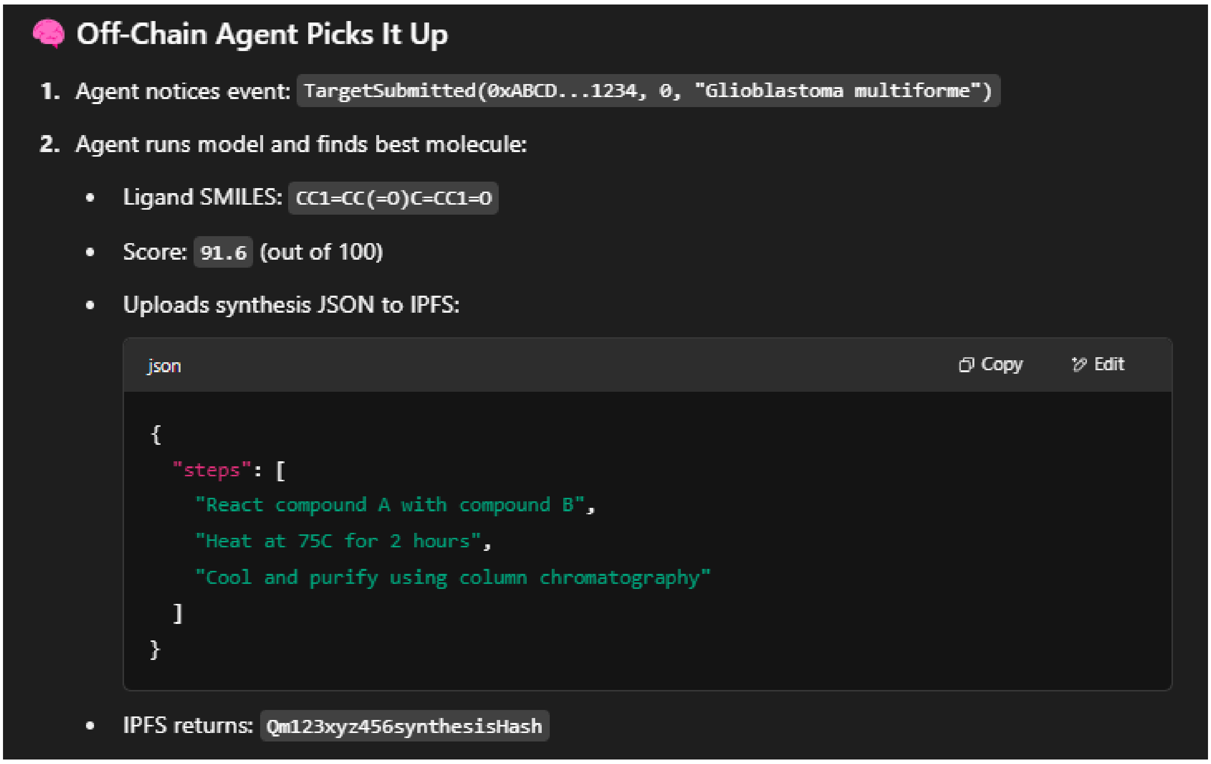
🧍 Researcher Submits a Target

```solidity
submitTarget("Glioblastoma multiforme")
```

⛓️ Onchain result:

```json
TargetSubmission {
  user: 0xABCD...1234,
  target: "Glioblastoma multiforme",
  timestamp: 1714500000,
  processed: false
}
```

Target ID = `0`

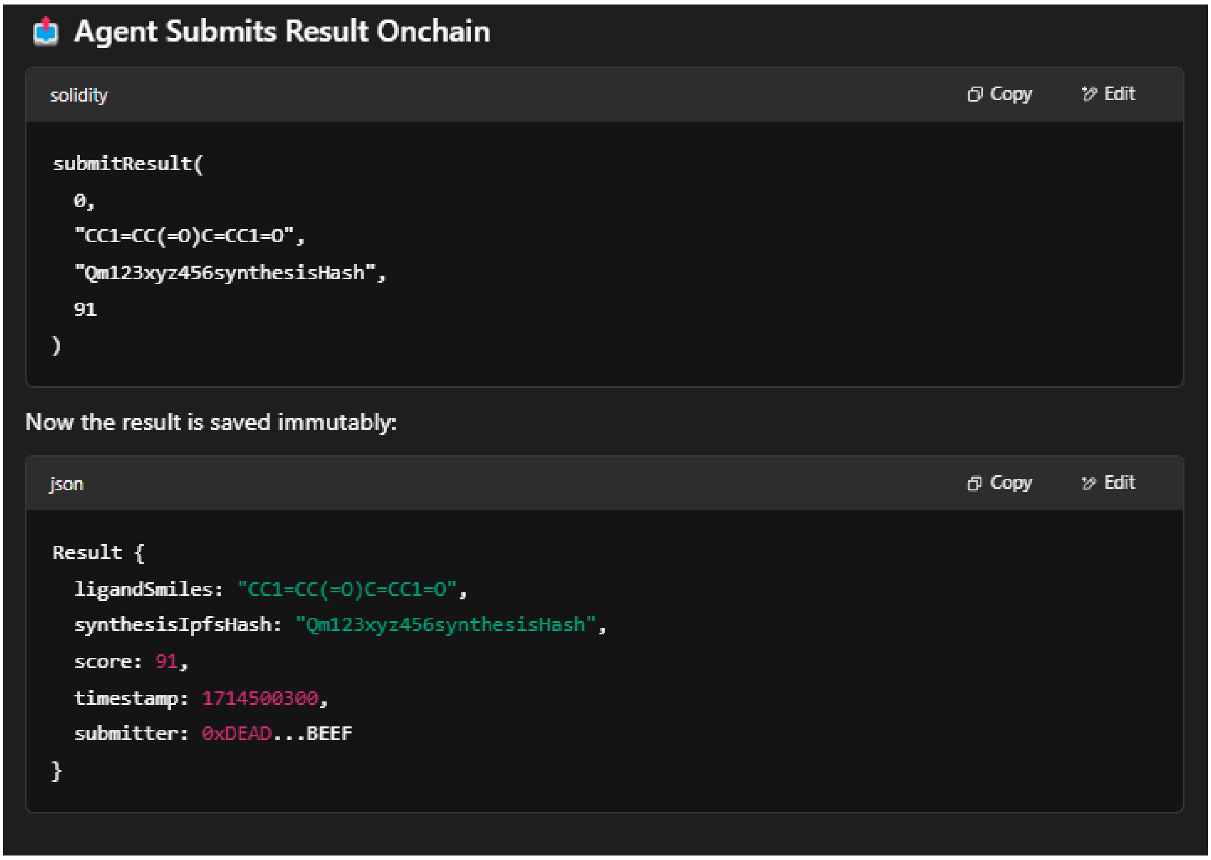


🧠 Off-Chain Agent Picks It Up

1. Agent notices event: `TargetSubmitted(0xABCD...1234, 0, "Glioblastoma multiforme")`
2. Agent runs model and finds best molecule:
   - Ligand SMILES: `CC1=CC(=O)C=CC1=O`
   - Score: `91.6` (out of 100)
   - Uploads synthesis JSON to IPFS:

     ```json
     {
       "steps": [
         "React compound A with compound B",
         "Heat at 75C for 2 hours",
         "Cool and purify using column chromatography"
       ]
     }
     ```

   - IPFS returns: `Qm123xyz456synthesisHash`

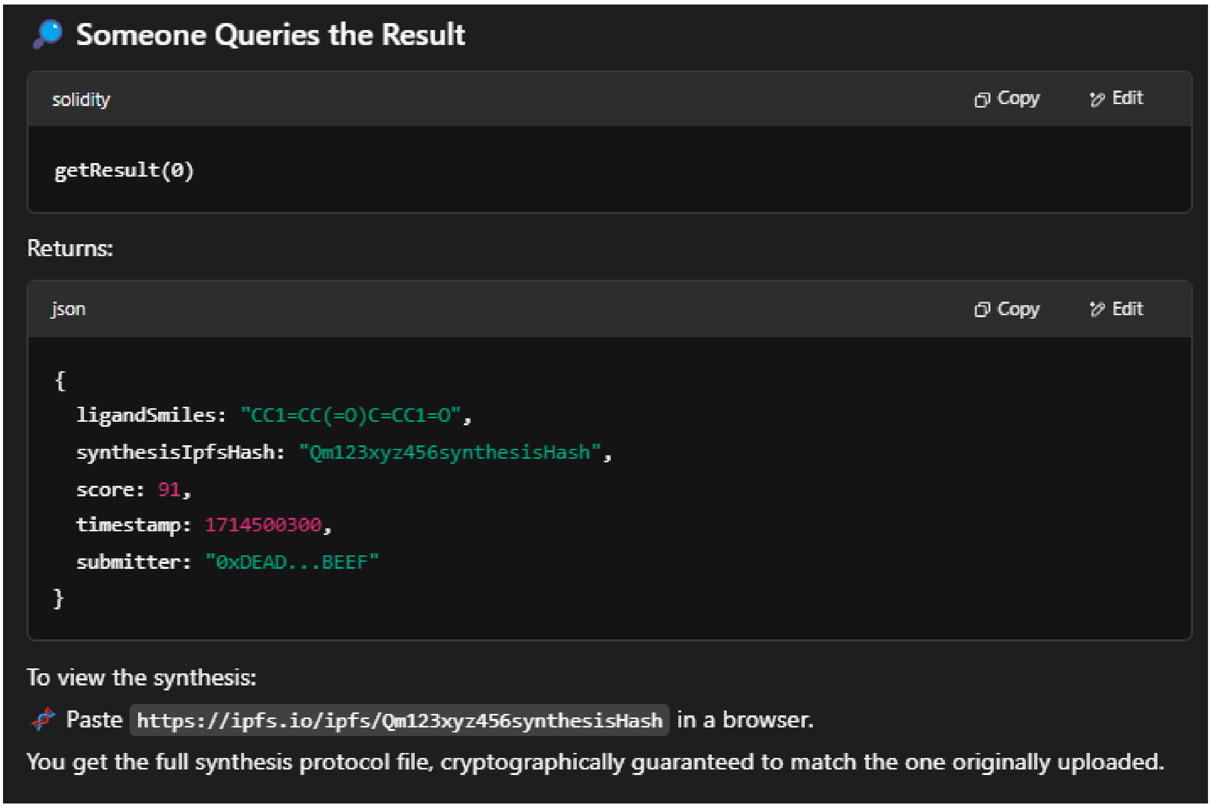


📤 Agent Submits Result Onchain

```solidity
submitResult(
  0,
  "CC1=CC(=O)C=CC1=O",
  "Qm123xyz456synthesisHash",
  91
)
```

Now the result is saved immutably:

```json
Result {
  ligandSmiles: "CC1=CC(=O)C=CC1=O",
  synthesisIpfsHash: "Qm123xyz456synthesisHash",
  score: 91,
  timestamp: 1714500300,
  submitter: 0xDEAD...BEEF
}
```



🔎 Someone Queries the Result

```solidity
getResult(0)
```

Returns:

```json
{
  ligandSmiles: "CC1=CC(=O)C=CC1=O",
  synthesisIpfsHash: "Qm123xyz456synthesisHash",
  score: 91,
  timestamp: 1714500300,
  submitter: "0xDEAD...BEEF"
}
```

To view the synthesis:

🧬 Paste `https://ipfs.io/ipfs/Qm123xyz456synthesisHash` in a browser.

You get the full synthesis protocol file, cryptographically guaranteed to match the one originally uploaded.

🧩 Automate Top Diseases Too

Running PolyKye **automatically on the top 100 diseases** would:

- Showcase the pipeline's power
- Seed the database with useful examples
- Prove that it works at scale

That can be the **"demo" layer** or foundation for public good — but the **real utility** comes when people start submitting unique, high-resolution questions.