Next Steps — `agentic-bo` Code Contributions

The engine is done. What's missing is the ability to **bring chemistry problems in** — molecules as SMILES strings, alloy compositions, reagent/solvent catalogs — and convert them into the clean tabular CSVs the engine already handles. The conversion needs to work **both ways**: encoding turns chemistry inputs into features HEBO can optimize over; decoding turns HEBO's suggested feature values back into something a chemist can act on. For numeric inputs (concentration, mass) decoding is trivial — read the value directly. For molecule inputs (ligand, solvent) decoding means a nearest-neighbor lookup to recover the original SMILES — no neural network needed, but it must be explicitly implemented.

Separately, the observer side needs maturing so a human (or another agent) can step in.

1 Work Areas

1.1 Observer API + Simulated Human-in-the-Loop

The `Observer` ABC is already in place. Extend it with:

- `SimulatedHumanObserver` — like `ProxyObserver` but with configurable noise and optional refusal (simulates a human declining an infeasible experiment). Useful for testing the full pipeline without real experiments.
- `AgentObserver` — delegates evaluation to a separate AI agent or tool-calling loop. The agent decides which experiments to run and returns observations. Keeps agent logic fully decoupled from the engine.

Note

The engine doesn't change — only new `Observer` subclasses are added on top of the existing `CallbackObserver` hook.

1.2 Conversion Layer — Chemistry → Feature CSV

Encoding is always required for this to work on problems beyond the 4 datasets. A new user brings molecules as SMILES strings, compositions, etc. — the engine can't eat those directly. The converter turns that into a feature CSV.

Decoding depends on what mode you're in:

Mode	Decode needed?	How
Library search (pick best from catalog)	Just a join	Encode preserves molecule IDs → HEBO suggests a row → look up original SMILES
Generative (HEBO invents new molecules)	True decoder	Latent vector → SMILES via VAE decoder — future work

For now (library search): the **non-negotiable design rule** is that every feature CSV must carry the original molecule identifier as a passthrough column. Without it, HEBO's suggestion is an uninterpretable vector of floats and you've lost the molecule.

Core pipeline:

```
user's molecules (SMILES / compositions / ...)
         |  [converter script]
feature CSV + molecule_id column (kept as passthrough)
         |  [bo_workflow.cli init]
HEBO runs -> suggests row
         |  join on molecule_id
original SMILES / name / formula <- chemist-readable result
```

Each converter lives in `scripts/converters/` and must:

1. Accept the raw chemistry input
2. Output a feature CSV the engine can ingest
3. **Preserve a molecule identifier column** so results are always interpretable

1.3 Specific Converter Scripts

1.3.1 3a. SMILES → RDKit Descriptors

- **Input:** CSV where some columns are numeric (concentration, mass, yield) and some are molecules (ligand, solvent, etc. as SMILES strings)
- **Process:** for each SMILES column, RDKit computes descriptors (MW, logP, TPSA, fingerprint bits, etc.) and replaces that one column with multiple descriptor columns. Numeric columns pass through untouched.
- **Output:** feature CSV ready for `bo_workflow.cli init --dataset ...`
- **Script:** `scripts/converters/smiles_to_descriptors.py`

```
Input: experiments.csv [concentration, mass, ligand_smiles, solvent_smiles, yield]
Output: features.csv   [concentration, mass, lig_MW, lig_logP, sol_MW, sol_logP, yield, ligand_smiles,
    solvent_smiles]
                        ^ pass through ^    ^ expanded descriptors ^        ^target ^ ID cols for decode ^
```

Decode: HEBO suggests a row → read concentration/mass directly → nearest-neighbor lookup on descriptor columns to find the original ligand SMILES → chemist gets an actionable experiment.

Note

RDKit produces ∼200 descriptors per molecule. The `build-oracle --max-features` flag handles selection, so no manual pruning needed.

1.3.2 3b. Composition/Simplex (for HEA)

- **Input:** CSV where element columns sum to 1 (e.g. `Fe`, `Co`, `Ni`, `target`)
- **Process:** Simplex → box reparameterization (see tutorial Appendix) so HEBO gets unconstrained $[0, 1]^{n-1}$ inputs
- **Output:** reparameterized feature CSV + a small JSON metadata file for interpreting results
- **Script:** `scripts/converters/composition_simplex.py`

Important

`HEA_alloy_data.csv` (currently open) needs this treatment — raw composition columns violate HEBO's independence assumption between parameters.

1.3.3 3c. Mixed-Variable Catalog (for BH synthesis)

- **Input:** CSV with categorical columns (reagent, solvent, base) + numeric amounts
- **Process:** already handled natively by the engine's `_infer_design_parameters` (it detects `cat` vs `num` columns automatically)
- **Output:** typically no conversion needed — just verify column types and hand to the engine
- **Script:** `scripts/converters/validate_mixed_catalog.py` (validation/sanity-check script rather than a full converter)

1.4 Skills Wiring (`.claude/skills/`)

For each converter, add a `.claude/skills/` entry that tells Claude Code:

- What kind of input this handles (SMILES, compositions, mixed catalog, etc.)
- Exact CLI invocation

- What the output CSV looks like
- Known caveats (e.g. “requires RDKit”, “simplex constraint must be validated first”)

Goal: Claude Code reads the dataset description and autonomously picks the right converter, runs it, then hands the output to `bo_workflow.cli init`.

2 Suggested Order of Work

1. **SimulatedHumanObserver**
2. **composition_simplex.py**
3. **smiles_to_descriptors.py**
4. **validate_mixed_catalog.py**
5. **Skills entries for each**

Start with (2) — pure math, no ML deps, and directly needed for `HEA_alloy_data.csv`.

Key principle: the abstraction can be unified later. Write concrete scripts first, find the common shape after.