zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

An Open-Source Data Infrastructure for Alloy Synthesis Research: Stage 1 --- Composition Screening and Literature Cross-Referencing

Iliya Radulov

October 2026

Abstract

Experimental materials-science labs routinely accumulate synthesis and characterization data across spreadsheets, instrument export files, and ad-hoc folders, with no durable link between a sample's recipe, its measured properties, and whether a similar composition has already been reported elsewhere. We present the first stage of an open-source data infrastructure built to close this loop for a single working alloy-synthesis lab: a relational schema for composition and synthesis records, a strict composition/mass calculator, three empirical alloy-design screening metrics (VEC, atomic-size mismatch, mixing enthalpy) together with a melt/boil-point synthesis-feasibility check, an automated cross-referencing layer against three public computational-materials databases (Materials Project, OQMD, Alexandria) with deduplication and adjustable relevance filtering, and a desktop GUI that ties the whole workflow to a single "new entry" action. The system is built and used by a practicing researcher rather than a software team, and we report it as a systems/methodology contribution: the architecture, the concrete engineering failures encountered while building it, and how each was diagnosed and fixed, rather than new materials-science results. We argue the specific defects found — a parser ambiguity that produced silent wrong answers, a partial code edit that broke a dependency chain invisibly, a database-metadata bug traceable to an un-cast numeric type, and a credential-hygiene gap that `.gitignore` alone did not close — generalize beyond this codebase to any small scientific-software project built by a domain expert without a software-engineering background. The system and the lessons from building it are reported here; two follow-on papers cover the validated characterization pipelines (Stage 2) and the retrieval-augmented synthesis advisor and CALPHAD thermodynamics module (Stage 3) built on top of this foundation.

1 Introduction

A working experimental alloy lab generates data at several disconnected points: a hand-written synthesis recipe, a diffractometer's raw output file, a magnetometer's hysteresis loop, a micrograph, and a researcher's memory of whether a similar composition has been tried before. In most small labs — including the one this system was built for — these pieces live in spreadsheets, instrument-vendor file formats, and loosely named folders, linked only by the researcher's own recall. This is adequate for a single project but breaks down precisely when it would be most valuable: when deciding whether a new composition is worth attempting, and when later trying to assemble a dataset for machine learning.

This paper reports the first stage of a response to that problem: an open-source data infrastructure built by a single materials-science researcher, without a software-engineering background, over several months of part-time development. Stage 1, reported here, establishes the core loop — composition in, screening metrics and literature cross-checks out, everything persisted to one queryable store

— and is deliberately scoped to stop there. Two later stages, each reported separately, extend it: Stage 2 adds validated pipelines that extract quantitative results from raw instrument files (XRD, VSM, SEM) and attach them to the same sample record, and Stage 3 adds a retrieval-augmented literature advisor and a CALPHAD-based thermodynamic feasibility check.

We deliberately frame this as a software-systems contribution rather than a materials-science one. The empirical screening metrics used (Section 4.3) are established in the high-entropy-alloy literature; the contribution is the infrastructure that makes them usable as a daily research habit rather than a one-off calculation, and the concrete, reproducible account of what went wrong while building that infrastructure and why. We believe this account is useful specifically because it comes from a domain scientist building research software without formal training in it — a situation common across experimental science, and one the software-engineering literature rarely documents from the inside.

Contributions.

1. An open-source, end-to-end data-capture architecture for alloy synthesis research, combining a composition calculator, physics-based screening metrics, and automated multi-database literature cross-referencing behind a single desktop workflow.
2. A deduplication and adjustable-relevance-cutoff scheme for reconciling literature-search results across three structurally different public materials databases.
3. A documented, reproducible account of five engineering findings — four defects in our own code (a silent parsing ambiguity, a dependency-breaking partial edit, a numeric-type database bug, and a credential-hygiene gap) and one dependency-level rejection (a third-party database client discarded after it was found to corrupt results silently) — each generalized into a lesson for similarly-resourced scientific software projects (Section 5).
4. A scoping discussion of what Stage 1 deliberately excludes, and how it hands off to the Stage 2 and Stage 3 systems built on top of it (Sections 6–7).

The rest of this paper is organized as follows. Section 2 positions the system relative to electronic lab notebooks and public materials-data repositories. Section 3 gives a system-level overview. Section 4 describes the architecture and implementation in detail. Section 5 reports the engineering defects found and the generalizable lessons drawn from them. Section 6 discusses scope and limitations, and Section 7 concludes.

2 Related Work and Positioning

Electronic lab notebooks (ELNs). General-purpose ELN platforms provide structured note-taking, file attachment, and search, but treat composition and screening as free text or custom fields rather than first-class, queryable, computed data. They do not natively compute VEC/delta/mixing-enthalpy from a formula string, nor cross-reference a new entry against external materials databases at entry time. The system described here can be read as a narrow, deeply specialized alternative to a general ELN for exactly one recurring task — "is this alloy composition already known, and is it synthesizable" — at the cost of being far less general.

Public computational-materials databases. Materials Project, OQMD, and Alexandria (Section 4.4) are themselves large, well-engineered systems for discovering and querying computed materials properties, each with its own API and client library. They are built for materials discovery — finding new candidate materials with target properties — not for a practicing synthesis lab's narrower question of whether a specific composition already under consideration is known and

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

how it might be made. This system does not reimplement or compete with them; it is a thin, opinionated orchestration layer that queries all three automatically for every new sample, normalizes their differing result shapes into one comparable structure, and removes the manual, error-prone, multi-tab-browser workflow that querying them individually would otherwise require.

High-entropy/multi-principal-element alloy design metrics. The VEC, δ, and ΔH_{mix} screening criteria implemented here (Section 4.3) are established empirical rules from the high-entropy-alloy design literature, used to estimate phase-formation tendency from composition alone before any synthesis attempt. Our contribution is not a new metric but making these metrics a zero-friction, always-computed part of every new sample entry, with full periodic-table coverage and an explicit failure mode (rather than a silently wrong number) when element data is incomplete.

Positioning. This system sits between a general ELN and a materials-discovery database: narrower than either, but built to make one specific, recurring researcher decision — should I attempt to synthesize this composition, and how — fast, consistent, and automatically cross-checked, in a tool scaled to a single lab rather than an institution.

3 System Overview

Infrastructure evolution and a recommended starting setup. The infrastructure behind this system was not designed up front; it grew iteratively, in step with the project's own requirements, through four distinct configurations: (1) a single laptop running the application, screening logic, and a local PostgreSQL instance together; (2) a more capable laptop, adopted once composition screening and multi-database literature queries made the original machine a bottleneck; (3) a split configuration, with the canonical database moved to an always-on home-network server while the laptop remained the interactive client, adopted once the database needed to be reachable consistently rather than only while the development laptop happened to be open; and (4) the current three-role configuration, in which a dedicated GPU-equipped desktop performs the heavier downstream computation introduced in later stages (local large-language-model inference for the Stage 3 RAG advisor, CALPHAD equilibrium calculations), the home server remains the sole canonical store for both the database and raw data files, and the original laptop now serves purely as a lightweight access point into the rest of the system.

We report this evolution rather than only a final architecture because we believe the lesson is in the trajectory, not just the endpoint. A deployment that intends to stay within Stage 1's scope — composition entry, screening, and literature cross-referencing for a single researcher — is well served by a single machine with a local PostgreSQL instance, and we do not recommend setting up distributed infrastructure prematurely. A deployment that anticipates growing toward the shared-access, characterization-pipeline, or compute-heavy components of later stages, however, is better served by establishing the three-role split — a thin access/development client, an always-on server as the single canonical data store, and a separate compute-capable machine for heavy processing — from the outset, rather than migrating mid-project as we did twice. Database layout (Section 4.1) and the server migration that made the shared-access role possible are reported in full in Stage 2.

Design goals. The system was designed around three goals, derived directly from the lab's prior pain points: (1) every new composition should be screened and cross-checked against prior art automatically, as a side effect of normal data entry rather than a separate manual step; (2) every record, once entered, should be permanently and queryably linked to its full context — synthesis route, screening result, literature-check outcome — rather than living as an isolated spreadsheet

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

row; (3) the system should degrade gracefully on incomplete or ambiguous input (missing element data, ambiguous formula strings, unreachable network services) by failing loudly and locally rather than silently producing a wrong answer or crashing.

Scope boundary. Stage 1, as reported here, ends at composition, screening, and literature cross-referencing. It accepts and stores raw characterization instrument files (XRD/VSM/SEM) for visualization and record-keeping — including sorting them by file type and sample ID (Section 4.5) — but does not yet extract quantitative results from them; that extraction pipeline, built and separately validated afterward, is Stage 2. It also does not yet include the retrieval-augmented synthesis advisor or thermodynamic phase-diagram tooling of Stage 3. This scoping was a conscious choice: each stage was brought to a stable, validated state before the next was started, rather than developing all three in parallel.

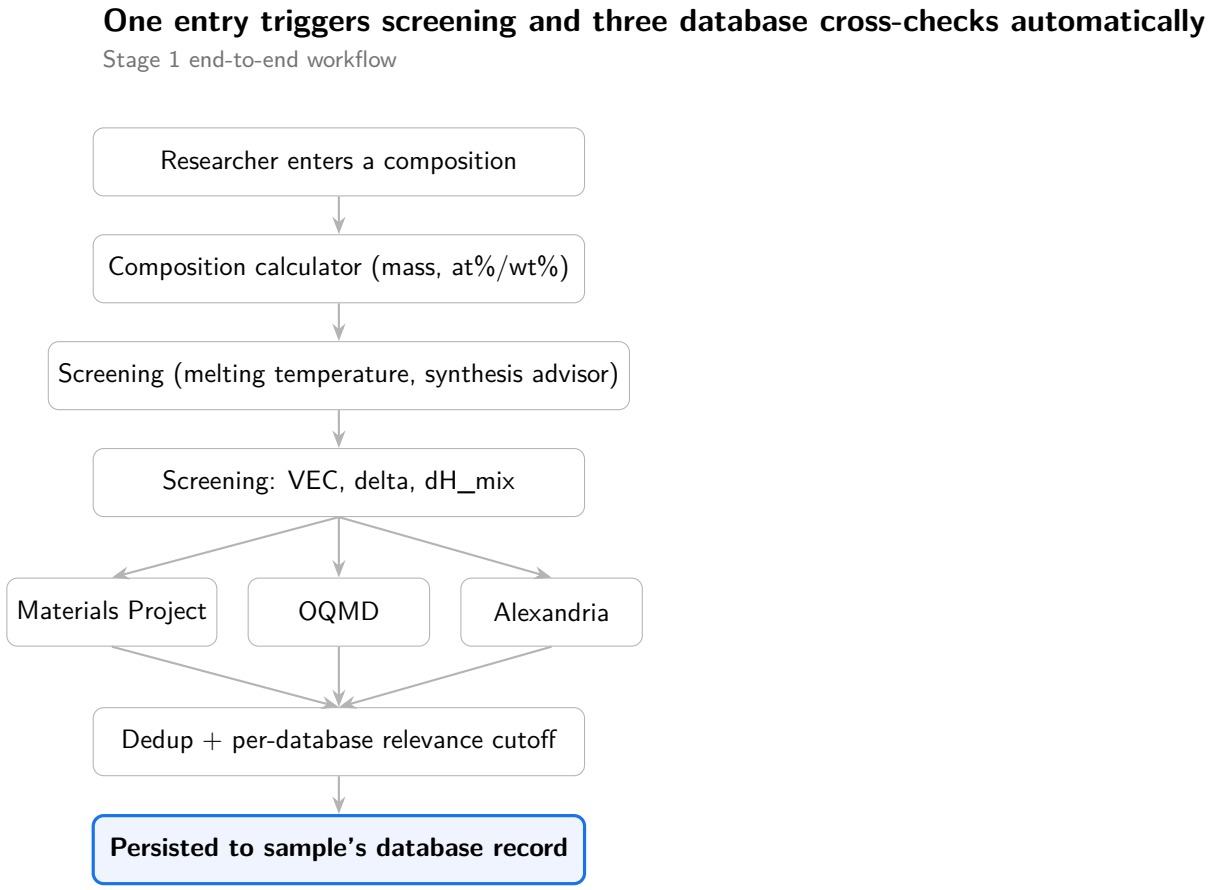


Figure 1: All three literature databases are queried in parallel for every new entry; results are then deduplicated and filtered before anything is written to the database, so the record a researcher sees already reflects the current relevance cutoff rather than a raw, unfiltered dump.

Figure 1 shows the Stage 1 workflow end to end, from a researcher entering a composition to a persisted, cross-checked database record.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

4 Architecture and Implementation

4.1 Database schema

The system persists to a PostgreSQL database with a `samples` table as its hub, holding composition (JSONB, for schema flexibility) alongside dedicated numeric columns for the three screening metrics so they remain directly indexable and filterable. Related tables cover `synthesis`, `characterization`, `properties`, `compositions` (normalized element fractions), `literature_sources`, `material_classes`, and `literature_checks` (one row per deduplicated candidate phase per source database per sample). Credentials are isolated in a `.env` file read via `python-dotenv`; the connection layer fails loudly rather than silently defaulting when a required variable is absent.

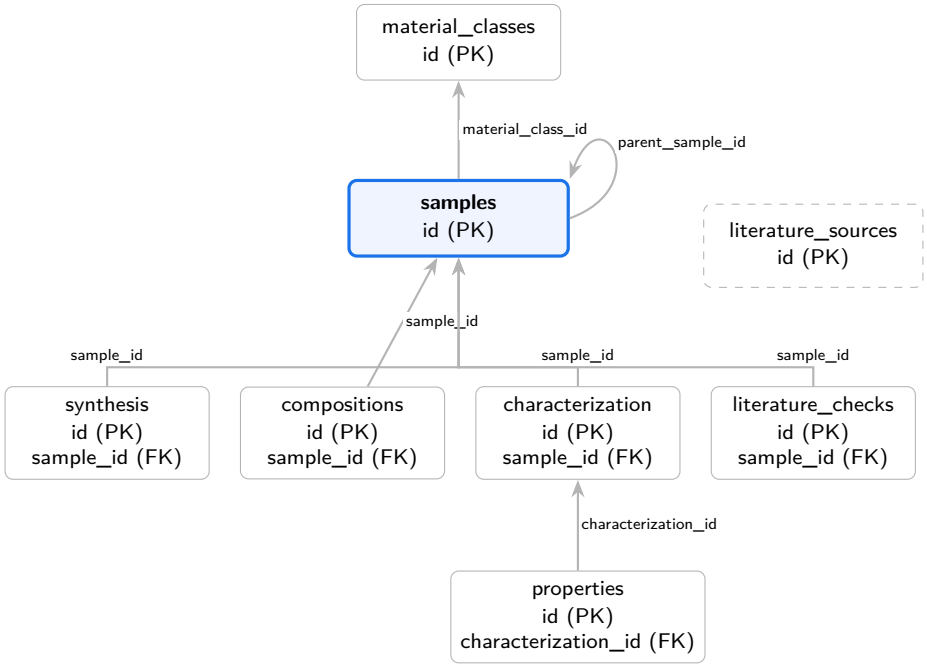


Figure 2: Core Stage 1 schema. `samples` is the hub table; `synthesis`, `compositions`, `characterization`, and `literature_checks` each reference it by `sample_id`, and `samples.parent_sample_id` self-references for derived/related samples. `properties` attaches to `characterization`. `literature_sources` (dashed) is populated with source DOIs from Materials Project's text-mined synthesis records when a sample is saved, but is not currently linked by a foreign key from `literature_checks`.

Figure 2 shows the core schema just described: `samples` as the hub, the four tables that reference it by `sample_id`, the self-referencing `parent_sample_id`, and `properties` attaching to `characterization`.

4.2 Composition calculator

The calculator parses a composition string, converts between atomic and weight percent, and computes synthesis masses. Its formula parser is deliberately strict: every element must be followed by an explicit numeric fraction (e.g. `Nd2Fe14B1`, never `Nd2Fe14B`). This was not the original design — an earlier, permissive parser treated a genuinely ambiguous input like "Cp" (which could mean

the elements C and P, or an unrecognized token) inconsistently rather than flagging it, a defect discussed in detail in Section 5.1. The calculator additionally supports two modes reflecting real synthesis practice: pre-alloy-based synthesis (starting from a commercial pre-alloyed ingot of known composition) and evaporation-loss compensation (an excess-fraction term for elements known to evaporate during melting), including an inverse mode that, given a fixed mass of pre-alloy on hand, solves for the raw element masses needed to reach a target composition while correctly separating contributions from elements that appear in both the pre-alloy and the raw additions.

4.3 Composition screening

Three metrics are computed for every composition: valence electron concentration (VEC) [1], atomic-size mismatch (δ) [2], and mixing enthalpy (ΔH_{mix}) [3,4], established empirical indicators of phase-formation tendency in multi-principal-element alloy design. VEC is computed as the composition-weighted mean of elemental valence electron counts; δ as the root-mean-square deviation of atomic radii from the composition-weighted mean; ΔH_{mix} via a pairwise Miedema-style parameter table [3,4]. VEC and δ draw on per-element physical constants spanning the full periodic table (103 elements). These metrics flag phase-formation tendency from composition alone; Section 4.4 narrows that estimate using existing literature, and the CALPHAD-based module reported in the Stage 3 paper narrows it further for compositions with a partially known chemical subsystem.

4.3.1 A silent data-table defect in ΔH_{mix}

ΔH_{mix} originally used a pairwise mixing-enthalpy table whose literature source was not recorded at implementation time, and cross-checking all 40 original pairs against Takeuchi and Inoue's tabulated Miedema-model values [3] (confirmed internally consistent with their earlier compilation of the same binary systems [4]) surfaced two independent problems. First, four pairs differed in sign — including the Fe–Nd pair, whose original value was opposite in sign to the tabulated one — a defect that would have silently stabilised compositions where the literature indicates destabilisation. Second, the regular-solution formula underlying these tabulated values,

$$\Delta H^{\mathrm{chem}} = \Sigma\, \Omega_{ij}\, c_i\, c_j, \quad \text{with } \Omega_{ij} = 4 \times \Delta H^{\mathrm{mix}}_{AB} \text{ [4]},$$

requires each binary value to be scaled by a factor of four before use as a pairwise interaction parameter, a scaling the original implementation omitted; the implementation also defaulted silently to zero for any pair absent from the table, masking missing coverage as a weak-interaction result — the same failure mode as the parser defect of Section 5.1, one layer deeper in the data pipeline.

The table and formula have since been replaced. ΔH_{mix} is now stored as the literature $\Delta H^{\mathrm{mix}}_{AB}$ value for each pair, expanded well beyond the original 40; all entries are from Takeuchi & Inoue 2005 [3], cross-checked entry by entry against a machine-readable copy of that table, and no entries remain flagged. The $\times 4$ regular-solution factor is applied explicitly inside the calculation and verified by a self-test confirming that an equiatomic binary reduces to exactly $\Delta H^{\mathrm{mix}}_{AB}$. An untabulated pair now raises an explicit error rather than defaulting to zero.

4.3.2 Incomplete-data failure modes

Two exceptions enforce the same design principle — a failure is loud and local, never a plausible-looking wrong number. `IncompleteElementDataError` is raised whenever a composition includes an element outside the tabulated element data; the newly added `IncompletePairDataError` is raised analogously whenever two tabulated elements lack a recorded ΔH_{mix} pair. Both the CLI

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

tool and the GUI catch these and degrade gracefully (screening is skipped for that run, with the affected columns left NULL) rather than either crashing or persisting a value that looks valid but was computed from an incomplete element or pair set. This is the same principle as the strict formula parser (Section 4.2) and the connection layer's refusal to silently default when a required environment variable is absent (Section 4.1): the only safe response to incomplete input is to refuse to compute at all.

4.3.3 Synthesis-feasibility check

A related but distinct check runs alongside the three metrics. `check_synthesis_feasibility` compares the melting and boiling points of the constituent elements, rather than estimating phase-formation tendency from composition. It finds the highest melting point and the lowest boiling point among the elements present and reports the margin between them, classifying the result as ok, caution, blocked, or unknown (when melt/boil data for an element is missing). A blocked composition — where an element would boil before another has fully melted under straightforward arc or induction melting — returns a list of suggested alternative routes (mechanical alloying by ball milling, powder sintering, diffusion bonding); a caution composition returns softer guidance (monitor mass loss during melting, check for literature precedent, run a small-scale test melt first).

Unlike VEC, δ, and ΔH_{mix}, this is not a phase-formation indicator from the high-entropy-alloy literature. It is a practical feasibility check on the synthesis route itself, reported here because it is the one screening output whose result is directly actionable — a blocked verdict names a specific alternative route, whereas the phase-formation metrics only narrow the space of compositions worth attempting. The distinction is deliberate: the three metrics answer "what phases might form," the feasibility check answers "can this be made by the route I have." The two together are what makes a new composition entry useful before any experimental work begins.

Figure 3 shows this in practice, on the New Entry tab: entering a composition ($W_{50}Zn_{50}$) triggers the composition calculator, the three screening metrics, and the feasibility check in one action, returning here a blocked verdict — Zn would be lost to vapor before W melts — together with the suggested alternative synthesis routes.

4.4 Multi-database literature cross-referencing

Every new composition is checked against three public materials databases before being logged:

Database	Access	Distinguishing data
Materials Project	Official `mp-api` client	DFT hull distance, text-mined synthesis recipes; oxide-heavy coverage
OQMD	OPTIMADE protocol	`experimentally_known` flag; no API key required
Alexandria	OPTIMADE protocol	Much larger (2.5M+ entries) computational dataset; no experimental flag

A fourth candidate, AFLOW, was evaluated and ultimately rejected for reasons that go beyond ordinary integration friction; see Section 5.5 for the full account of why it was dropped and what that implies more generally about adopting unmaintained scientific-software dependencies.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

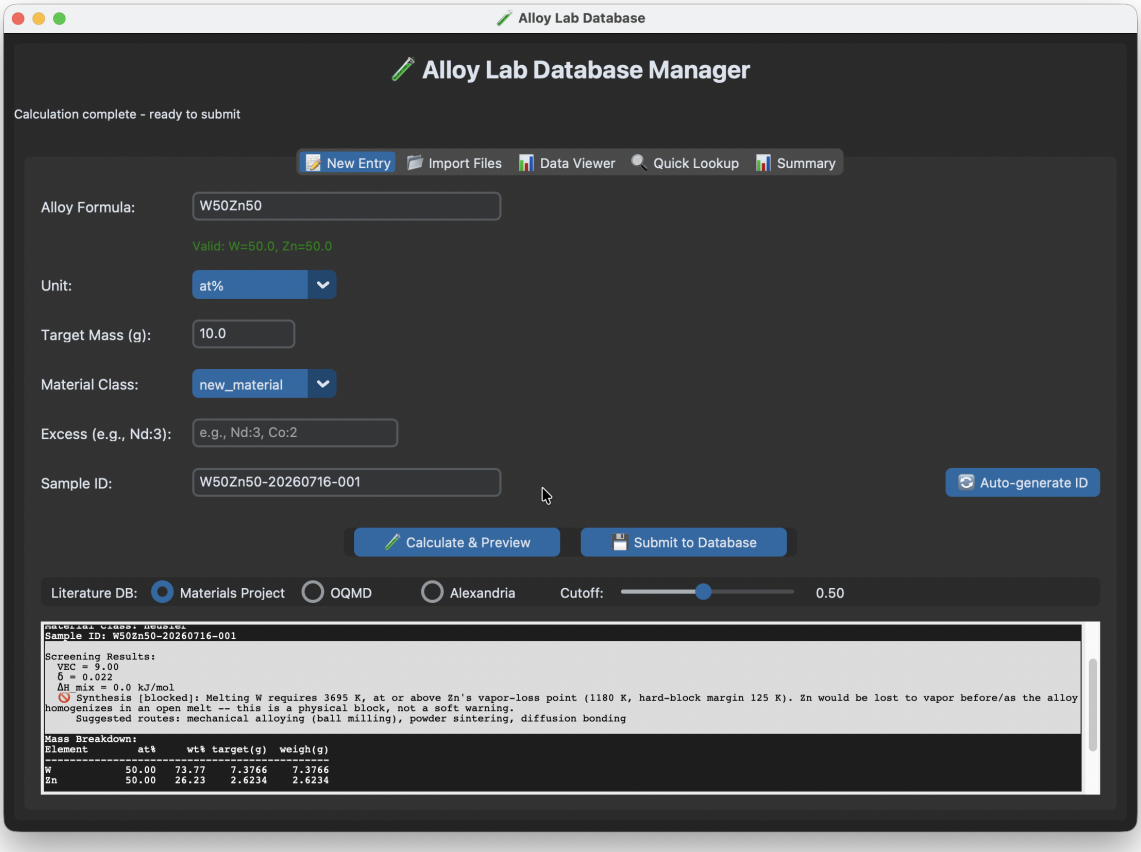


Figure 3: The New Entry tab after “Calculate & Preview”: composition parsing, mass breakdown, the three screening metrics, and a blocked synthesis-feasibility verdict with suggested alternative routes, all from a single action.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

4.4.1 Deduplication and relevance filtering

The three working sources are normalized through a shared adapter layer (see Section 4.5 for how results are then surfaced to the researcher) into one common candidate shape, then processed by two stages: deduplication by formula (preferring experimentally-known results, then most stable, then closest composition match), and a per-database, user-adjustable distance cutoff — added after real use showed Alexandria's much larger dataset returning dozens of only loosely related formulas per query, more volume than deduplication alone resolves. Both the OQMD and Alexandria clients carry custom retry logic for transient server errors, since the underlying OPTIMADE client library's built-in retry only triggers on HTTP 429 and otherwise surfaces a failure without retrying.

4.4.2 Four-tier literature-match classification

Results are additionally classified into a four-tier match system: an exact match (tier 1) or a similar composition (tier 2) are read directly off the returned entries; a partial subsystem match (tier 3) — where only part of the candidate's chemical system is covered by existing literature — is where a subsequent CALPHAD-based thermodynamic check (reported in the Stage 3 paper) adds the most value, since the database result alone cannot resolve phase behavior for the untested remainder of the composition; nothing found (tier 4) hands the full composition to that same experimental-plus-modeling cycle with no literature prior to lean on. The tier assigned at this stage is therefore not just a novelty read but a direct input to how much downstream thermodynamic modeling a candidate composition warrants.

4.5 Desktop GUI and data-sorting utility

The primary researcher-facing interface is a `customtkinter` desktop application structured around five tabs (new entry, file import, data viewer, quick lookup, summary). All three literature databases are queried upfront on a single "Calculate & Preview" action and cached locally; independent, per-database relevance-cutoff sliders then re-filter the cached results without any further network call, keeping the interactive exploration step fast regardless of how many times the cutoff is adjusted. A separate, narrower command-line tool remains available for fast, scriptable data import when the GUI's interactivity is unnecessary — a deliberate CLI/GUI split, not legacy duplication: the CLI stays dialogue-only for scripted import from existing files, while anything visual or exploratory goes through the GUI.

Figure 4 shows the Data Viewer tab rendering a raw XRD pattern. Consistent with the scope boundary set out in Section 3, Stage 1 stops at this visual inspection step: it does not extract peak positions, phase identification, or a lattice parameter from the pattern, since that quantitative extraction is the Stage 2 characterization pipeline.

A standalone utility separately auto-classifies files dropped into an unsorted working folder, by both file type (XRD/SEM/VSM/ICP/process notes/images, etc.) and sample ID, using extension- and filename-pattern heuristics — reducing a recurring manual-triage step common to any lab with heterogeneous instrument output to a single command. Figure 5 shows the Import Files tab after a batch import: files auto-detected in a working folder, matched to sample IDs, and queued for import rather than sorted and renamed by hand.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

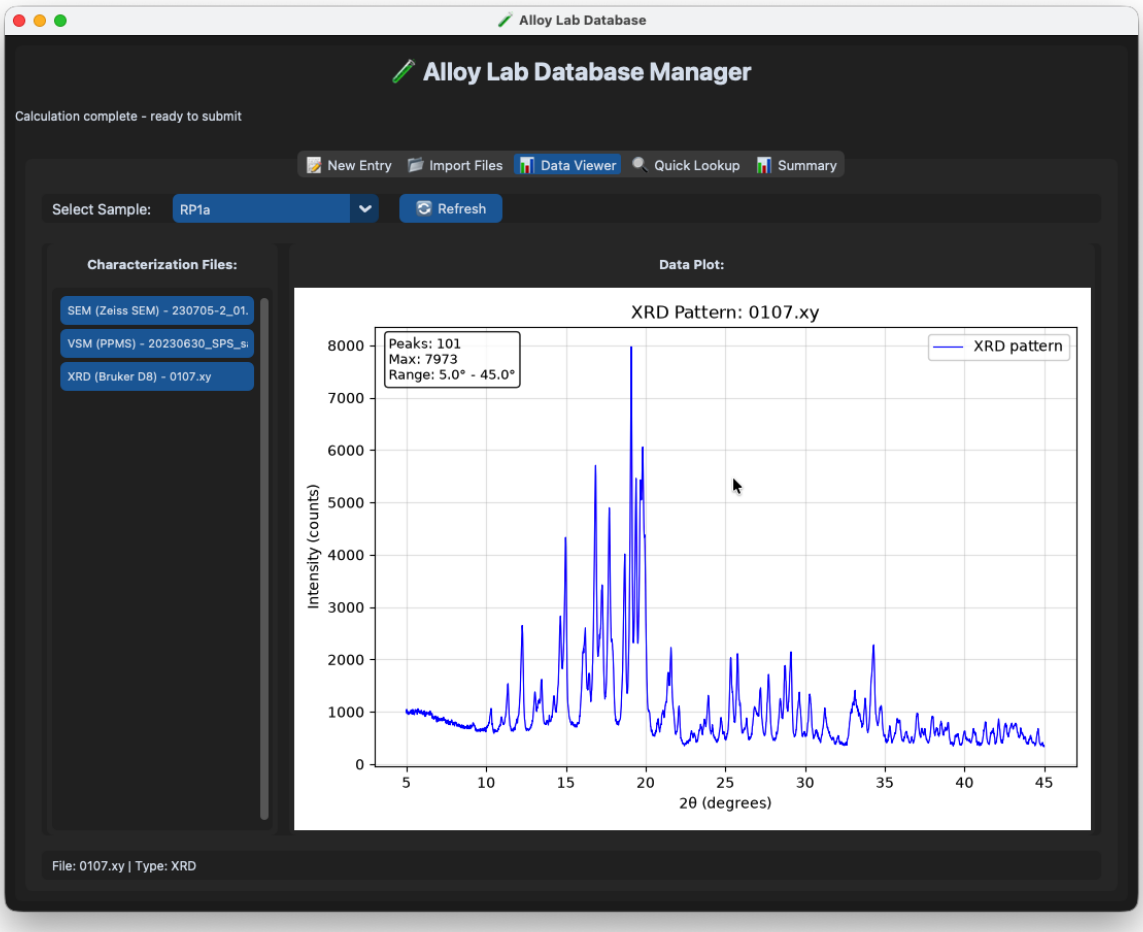


Figure 4: The Data Viewer tab displaying a raw XRD pattern (sample RP1a, Bruker D8): peak count, maximum intensity, and angular range are shown for visual inspection, with no quantitative extraction performed at this stage.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

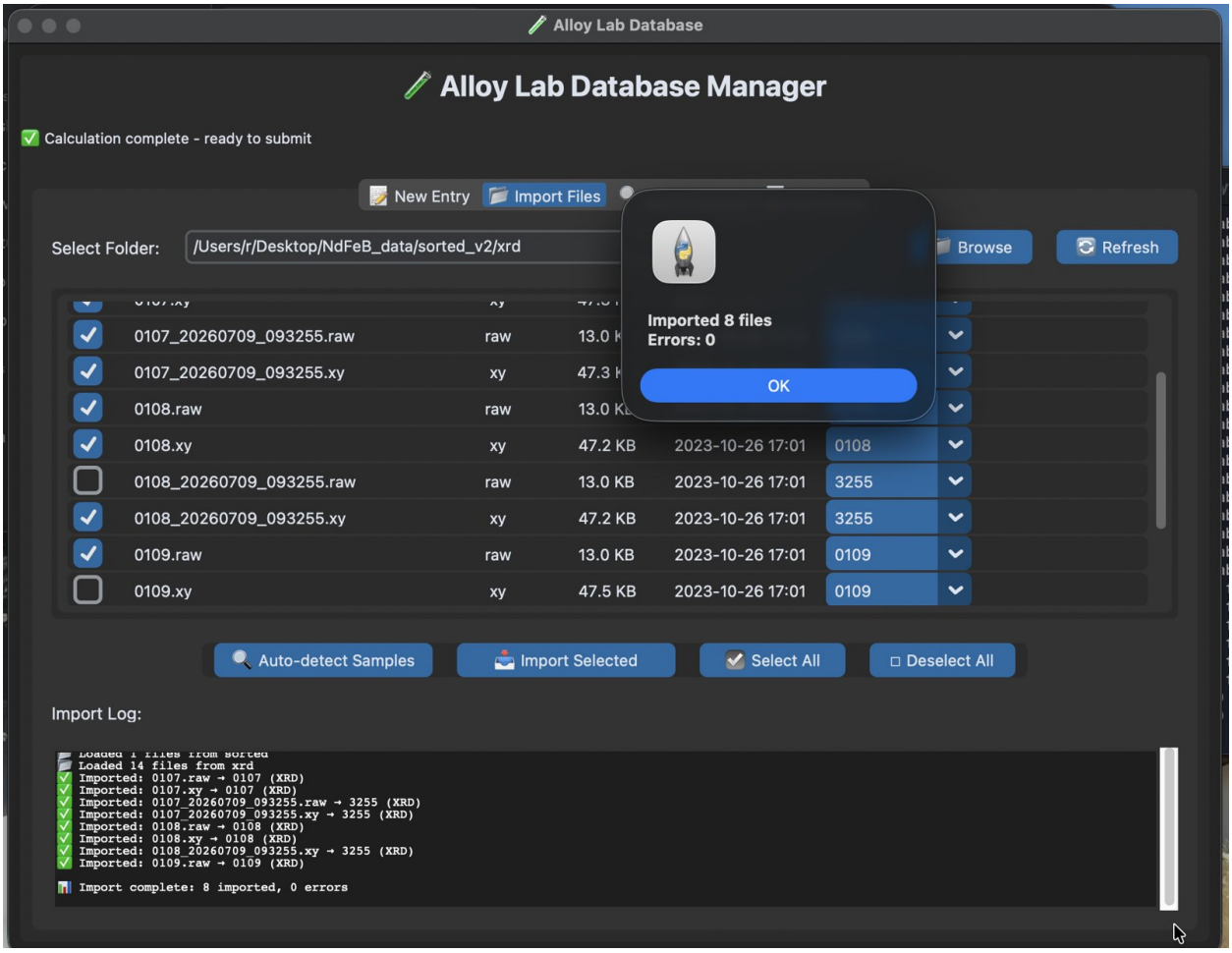


Figure 5: The Import Files tab: raw instrument files auto-detected in a working folder, matched to sample IDs, and imported in a batch, with a running log of what was imported and any errors.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

5 Engineering Validation and Lessons Learned

We report five engineering findings in detail — four defects in our own code and one dependency-level rejection — each chosen because it generalizes beyond this codebase to any small scientific-software project built by a domain expert without formal software-engineering training, a situation we believe is common and underreported.

5.1 Silent parsing ambiguity

Defect. An early version of the composition parser treated a string like "Cp" inconsistently: C and P are both real element symbols, so "Cp" is a genuine ambiguity (is it one element, Cp — not a real element — or two, C and P?), not a malformed input. A permissive parser resolved this silently, with no indication to the user that a judgment call had been made.

Fix. The parser was rewritten to be strict: every element symbol must be immediately followed by an explicit numeric fraction. This converts every ambiguous case into an explicit, user-visible parse error rather than a silent, possibly-wrong interpretation.

Generalizable lesson. A parser's job in scientific software is not just to accept valid input but to actively detect when a domain-specific ambiguity exists; permissive parsing that "just works" on the common case is the more dangerous failure mode, because the single output looks exactly as trustworthy as a correct one.

5.2 A partial edit that broke a dependency chain invisibly

Defect. An edit adding a new distance-cutoff filtering function accidentally deleted an existing helper function's signature while leaving its body behind as orphaned, syntactically valid dead code nested inside the wrong function. This silently broke literature lookups across all three database integrations simultaneously (`NameError: name '_is_better' is not defined`), but the defect shipped because the fix was verified by testing only the newly added function in isolation — not the full call chain a real bug report would exercise.

Fix. The orphaned code was restored as its own properly-scoped function, and the fix was this time verified by exercising the actual, complete call chain (`MatchResult` → `from_mp_results` → `dedup_by_formula` → `filter_by_distance` → `GUI render`) rather than the single function that had been directly edited.

Generalizable lesson. Verifying a fix against the function that was edited, rather than the full path a bug report actually came from, systematically under-tests exactly the kind of regression most likely in small, single-maintainer codebases: a change whose damage is in what it silently deleted elsewhere, not in what it added.

5.3 A numeric-type bug surfaced as a database error

Defect. A computed lattice parameter (from the Stage-2 XRD pipeline, built immediately after this one and sharing its persistence layer) was inserted into the database via string formatting, which let a NumPy scalar's internal string representation leak literally into the generated SQL text rather than being converted to a native type first — producing a database error (`schema "np" does not exist`) with no obvious connection to its numeric-type root cause.

Fix. Explicit casting to a native Python `float` before binding the value as a parameterized query argument.

Generalizable lesson. Numeric library types (NumPy scalars, pandas values) are not safely interchangeable with native types across a string-based interface boundary; an error message at that boundary (here, a database driver) can point entirely away from the actual root cause, which is worth deliberately checking for whenever a numeric value crosses from a scientific-computing library into a persistence or serialization layer.

5.4 A credential-hygiene gap `.gitignore` alone did not close

Defect. After migrating the database password out of source code and into a `.env` file, a commented-out (not deleted) copy of the plaintext password was found still present in a configuration file on a later review — missed by an initial grep pass that searched only for active, uncommented occurrences. Separately, a data folder containing raw experimental files was added to `.gitignore` only after it had already been committed; since `.gitignore` has no retroactive effect on already-tracked files, the folder remained fully present in git history despite appearing absent from the current working tree.

Fix. The commented-out password was deleted and git history was audited directly (`git log --all -p | grep <string>`) rather than trusting current file contents, to confirm the plaintext value had never actually been committed. The data folder was removed from history with `git-filter-repo`, verified first on a disposable clone (confirming the path no longer appeared anywhere in history and that repository size dropped accordingly) before the same procedure was applied to the real repository.

Generalizable lesson. Neither a comment-out edit nor a `.gitignore` entry is itself a security control; the only way to verify whether a secret or sensitive file was ever exposed is to audit git history directly, not current file contents, and a repository should be set private while that audit is pending rather than after.

5.5 A dependency rejected for silent-corruption risk

Context. AFLOW was the fourth materials database considered for the literature cross-referencing pipeline described in Section 4.4, alongside Materials Project, OQMD, and Alexandria. Its official Python client (version 0.0.11) had not been meaningfully maintained since roughly 2017–2019.

Findings. Three independent problems surfaced during integration, each sufficient on its own to warrant caution: the client's query endpoint returned a live HTTP 404, indicating a moved or retired server-side endpoint rather than a transient outage; the underlying AFLUX query language has an unresolved AND/OR precedence ambiguity in its filter syntax, which the client does not disambiguate or warn about; and, most seriously, the client's keyword-filter objects are implemented as mutable singletons, so a second `.filter()` call silently mutates and corrupts the filter state left behind by a first call — meaning two sequential, independently-reasoned queries in the same process can silently contaminate each other's results, with no exception raised and nothing in the returned data to indicate anything went wrong.

Decision. The client was dropped rather than patched or worked around. A patched fork would have meant maintaining a fix for an abandoned upstream project indefinitely; a workaround (a fresh filter object per query, say) would have papered over a defect class — silent mutation of shared state — that could plausibly resurface elsewhere in the same dependency. Alexandria was adopted

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

in AFLOW's place and, independently of this decision, turned out to better fill the role of a large, purely computational third database (Section 4.4).

Generalizable lesson. An unmaintained dependency's risk is not fully captured by whether it currently runs; a confirmed silent-corruption bug in shared, mutable state is a standing risk to every future use of that dependency, not a one-time defect to patch around, and is reason enough to reject it even when a workaround for the immediate symptom is available.

5.6 Summary

All four code defects share a structure: each produced an output that was plausible-looking rather than obviously broken, and each was caught by exercising the system under real or realistically adverse conditions (a genuinely ambiguous real formula, a real downstream bug report, a real database insert, a real git-history audit) rather than by code review or an isolated unit test. The dependency rejection of Section 5.5 exhibits the same failure mode one layer removed — a silent corruption whose output looks as trustworthy as a correct one — and was caught by the same kind of real-use exercise, in this case an integration attempt rather than a bug report. We take this as the central practical lesson of Stage 1: for a small scientific-software project with no dedicated QA process, validating against real use is doing a substantial share of the work that a larger project's test suite and code review would otherwise do, and that validation has to reach the full path a defect could plausibly take, not just the code that was most recently touched.

6 Discussion and Limitations

What this paper deliberately does not report. Consistent with the software-architecture framing of this work, we do not report specific sample compositions, measured properties, or literature-match outcomes for named samples. Where an example composition is needed to illustrate a design choice, it is representative or synthetic rather than drawn from an actual unpublished sample. This is a reporting choice, not a confidentiality constraint — the underlying data comes from open, previously published projects — made because the contribution of this paper is the infrastructure and the engineering process, not the materials-science findings it produces.

Single-lab, single-maintainer scope. The system was built by and for one researcher's own lab. It has not been evaluated across multiple labs, multiple maintainers, or at a scale beyond roughly 50 samples at the time Stage 1 closed. We therefore make no claim that the specific architectural choices here (a single PostgreSQL instance, a desktop GUI as the primary interface, JSONB for composition storage) would scale unchanged to a multi-user or multi-lab deployment; they were right-sized for the problem actually being solved.

No formal test suite. The system has no automated test suite in the conventional sense; validation instead relied on the kind of real-use exercising described in Section 5. We view this as an honest account of how a time-constrained, single-maintainer scientific-software project actually gets validated in practice, not as a recommended substitute for automated testing where the resources for it exist.

Database selection is not exhaustive. Three materials databases were integrated and a fourth (AFLOW) was evaluated and rejected (Section 5.5); this is not a claim that no other public materials database would be worth integrating, only that these three covered the lab's practical needs at acceptable engineering cost.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

`literature_sources` is not yet cross-linked. The `literature_sources` table (Figure 2) is populated with source DOIs from Materials Project's text-mined synthesis records when a sample is saved, but is not yet linked by a foreign key from `literature_checks`: Stage 1 only needs the per-check match classification (Section 4.4.2), not a durable, queryable link to a specific stored literature record. That link is expected to be added in Stage 3, once the retrieval-augmented synthesis advisor needs to query stored literature entries directly rather than treat each database check as a one-off lookup.

Relationship to Stage 2 and Stage 3. This paper reports Stage 1 in isolation because it reached a stable, validated state before Stage 2 development began, and because each stage's validated scope is large enough to report independently. Stage 2 (validated XRD/VSM/SEM characterization pipelines, an element-fraction table, and a multi-machine server migration) and Stage 3 (a retrieval-augmented literature advisor and a CALPHAD thermodynamic feasibility module) are built directly on top of the schema and conventions established here, and are reported as separate papers.

A snapshot, not a live document. This paper describes the Stage 1 system as a fixed snapshot, frozen once Stage 1 reached the stable, validated state reported here. The underlying codebase is a living project that continues to develop after this report, and the repository is organized precisely so that this snapshot stays accurate: each stage keeps its own subfolder, left untouched once its paper is written, while ongoing work — including the shared root utilities all stages build on — continues elsewhere. Where the current system differs from what is described here, the difference should be in a positive direction (a fix, an added capability, broader test coverage) rather than a regression, since that shared foundation keeps improving as later stages are built on top of it. Readers should treat this paper as a dated account of the system at the point Stage 1 closed, not as live documentation of its current state.

7 Conclusion and Future Work

We presented Stage 1 of an open-source data infrastructure for alloy synthesis research: a composition calculator, physics-based screening metrics, automated multi-database literature cross-referencing, and a desktop GUI, built end to end by a single domain-expert researcher without formal software-engineering training. We reported the system's architecture and, in detail, five engineering findings — four defects in our own code (a silent parsing ambiguity, a dependency-breaking partial edit, a numeric-type database bug, and a credential-hygiene gap that `.gitignore` alone did not close) and one dependency-level rejection (a third-party database client discarded for silent-corruption risk, Section 5.5) — each generalized into a lesson we believe applies broadly to small, similarly-resourced scientific-software projects: that validation has to exercise the full path a defect could plausibly take, and that a config-file comment-out or a `.gitignore` entry is not itself a security control.

Two follow-on papers extend this work directly. Stage 2 reports validated, instrument-specific characterization pipelines (XRD, VSM, SEM) that attach quantitative measurements to the sample records this system creates, an element-fraction table enabling machine-learning-style queries, and a migration of the database to shared, always-on infrastructure. Stage 3 reports a retrieval-augmented synthesis advisor built on a large materials-science literature corpus and a CALPHAD-based thermodynamic phase-diagram module for pre-synthesis feasibility screening. Together, the three stages are intended to demonstrate that a practicing researcher can build durable, reusable research infrastructure incrementally, validating and reporting each stage before the next begins, rather than deferring all engineering investment until a larger, better-resourced project makes it possible.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.

References

[1] Guo, S., Ng, C., Lu, J., & Liu, C. T. (2011). Effect of valence electron concentration on stability of fcc or bcc phase in high-entropy alloys. *Journal of Applied Physics*, 109(10), 103505.

[2] Yang, X., & Zhang, Y. (2012). Prediction of high-entropy stabilized solid-solution in multi-component alloys. *Materials Chemistry and Physics*, 132(2–3), 233–238.

[3] Takeuchi, A., & Inoue, A. (2005). Classification of bulk metallic glasses by atomic size difference, heat of mixing and period of constituent elements and its application to characterization of the main alloying element. *Materials Transactions*, 46(12), 2817–2829.

[4] Takeuchi, A., & Inoue, A. (2000). Calculations of mixing enthalpy and mismatch entropy for ternary amorphous alloys. *Materials Transactions, JIM*, 41(11), 1372–1378.

zenodo:23156552 [cond-mat.mtrl-sci] 5 Oct 2026 DOI: 10.5281/zenodo.23156552.