ChatGPT

# Alphamole Stability Prediction: Feature and Pipeline Prioritization

## 1. Inputs to the Neural Network

**1. Element count vector (composition) – [2].** The simplest input is a fixed-length vector of element counts (e.g.

$$H, C, N, O, \ldots$$

) obtained by parsing the formula. This directly encodes stoichiometry (e.g. $H_2$ → [2,0,0,0], CO → [0,1,0,1]) and is trivial to compute. It captures basic composition differences (e.g. $H_2$ vs HO) which strongly influence stability (e.g. $H_2$ is stable, HO is a radical). This applies to any molecule size (N=2,3,4…) since it generalizes to all formulas. *(Feature source: chemical rules (2).)*

**2. Total atom count (N) – [2].** Simply summing the element counts gives the total number of atoms (the size of the molecule). In phased models the network may be separate by N, but in a combined model an explicit feature for N helps the network distinguish diatomic vs triatomic chemistry. It's trivial to compute but provides a basic signal (larger N means more possible bonding complexity). For N=2 it is constant (2), but for mixed training (N=2–4) it disambiguates molecule size. *(Feature source: chemical rules (2).)*

**3. Total valence electron count – [2].** Sum the valence electrons for each atom (from a static lookup table) weighted by their counts. For example, $H_2O$ has 2·(1) + 1·(6) = 8 valence electrons. Compounds with a "closed shell" (even total valence electrons satisfying octets/duplets) are more likely stable. In practice, an even total often correlates with non-radical species, whereas an odd total often indicates radicals (which are usually unstable). This feature helps capture electron-count rules across N. *(Based on octet chemistry, which is a chemical rule* [1] *.)*

**4. Aggregate electronegativity features – [2].** Compute properties like the average electronegativity of all atoms and the maximum electronegativity difference present. Electronegativity is a fundamental property that measures an atom's ability to attract electrons [2] . For example, large electronegativity differences in a molecule indicate ionic bonding tendencies (e.g. NaCl), while small differences suggest covalent bonds. By including e.g. "mean electronegativity" and "max–min electronegativity" in the feature vector, the network can infer bond polarity and likely bond types. These are easily computed from known elemental values (Pauling scale) and apply whenever $\geq$2 different elements are present. *(Feature source: chemical rules (2).)*

**5. Atomic property aggregates (e.g. molecular weight, atomic radii) – [2].** Other simple aggregations include the sum of atomic weights (molecular mass) and averages of atomic radii or group numbers. For instance, heavier (larger) atoms often form weaker bonds, which may influence stability. Such features are easy to compute from periodic table data. They give the model basic physical context (e.g. MW distinguishes $H_2$ vs $O_2$). These properties have been used in materials ML – for example, a state-of-the-art materials graph

network used atomic covalent radii, group number, and valence electrons as input features [3]. Here they serve as continuous features summarizing composition. *(Feature source: chemical rules (2).)*

**6. Bonding heuristics (advanced) – [2].** More complex features might estimate how many bonds the molecule can form. For example, one could compute the "electron pair bond count" as half the total valence electrons, or infer if multiple bonds are needed for octets. Another heuristic is the octet/duplet satisfaction: for each atom, how many electrons needed to fill its valence shell, and compare to actual. These calculations are nontrivial (especially for N>2) but can be automated by rule-based code. Such features directly encode chemical bonding constraints (e.g. C typically forms 4 bonds, O 2 bonds, H 1 bond). They would apply more strongly for small N where Lewis structures can be exhaustively checked. We rank these as hardest because they require implementing chemical logic, but they provide deep insight into stability. *(Feature source: chemical rules (2).)*

## 2. Data Preprocessing

**1. Formula parsing and vectorization (easy).** The raw chemical formula (e.g. "$CO_2$") must be parsed into the numerical feature vector (element counts, valence sum, etc.). This step involves tokenizing element symbols and counts (e.g. regex or a chemistry parser). It is necessary to convert human-readable formulas into numeric arrays. Complexity is low (O(length of formula)), but careful coding is needed to handle multicharacter symbols and implicit 1's.

**2. Feature encoding (easy).** All numerical features (counts, totals, etc.) must be consistently ordered and formatted. If any feature is categorical (e.g. element identity), one-hot encoding is used [4]. In our case, the element-count vector itself is like a one-hot for each element count. This ensures the neural net can ingest features correctly. Encoding is typically trivial with libraries (pandas, sklearn). Importantly, chemical features like electronegativity are continuous and can be used directly.

**3. Feature scaling/normalization (easy).** Numeric features often have different scales (e.g. atomic count vs. electronegativity). Standardization (zero mean, unit variance) or min–max scaling ensures no single feature dominates learning. For example, sklearn's StandardScaler can transform features so that each has mean$\approx$0 and variance$\approx$1 [5]. This is important for neural nets, which converge faster when inputs are normalized. Implementation is straightforward (compute mean/std on training set, apply to all).

**4. Balancing and splitting data (moderate).** Stability labels may be imbalanced (possibly far more unstable formulas than known stable ones). To avoid bias, techniques like oversampling the minority class or undersampling the majority can be used [6]. For instance, one can randomly duplicate stable examples or apply SMOTE. Alternatively, using class-weighted loss in training handles imbalance. This preprocessing step is moderately complex to implement but crucial: class imbalance can make models ignore the rare "stable" class. After balancing, we also randomly split into training/validation/test sets to evaluate performance reliably.

**5. Feature filtering or augmentation (hard).** Depending on results, one might remove poorly predictive features or create new interaction features. For example, converting counts to fractions (elemental percentages) might help the model focus on relative composition. Data augmentation (e.g. permuting order of atoms, though formula order is canonical) is usually not needed since representation is permutation-invariant. These steps add complexity and are done iteratively based on model feedback.

## 3. Labeling Confidence

**1. PubChem/Database presence (1).** The simplest way to label a formula as "stable" is to check if it exists as a known compound in a database like PubChem. If an exact formula (and even better, a specific structure) appears in PubChem, that is strong empirical evidence of stability. Implementation involves querying the PubChem API for the formula; if found, assign label=1. This method contributes direct empirical validation. Its reliability is high for known compounds, but absence of a hit does *not* prove instability (the database is large but not exhaustive). It is easy to implement (1) and provides a primary ground truth for many cases.

**2. Valence/OCTET rule scoring (2).** A rule-based approach is to check electron counts: for each atom, can it achieve a full valence shell given the formula? For example, sum the total valence electrons and see if electrons can be paired to give each (non-H) atom an octet [1]. If the octet (or duplet for H/He) can be satisfied without leftover electrons, we score the formula as likely stable (label=1). Implementation is moderate: one must compute needed vs available electrons for each atom. This approach can catch radical cases (odd electron counts) and hypervalent possibilities. Its reliability is chemical (2); many stable molecules obey the octet rule, but there are exceptions (e.g. NO, $PF_5$). As a "confidence score," one could use a binary or soft flag from this rule to augment the database label.

**3. Manual expert overrides (1).** A chemist's intuition can override or label edge cases. For example, a human might know that "$C_3$" (carbon trimer) is not a stable neutral molecule, whereas "$CH_3^+$" is stable. We can encode a list of special cases or apply simple rules (e.g. "no molecule with only helium atoms except $He_2$ under extreme conditions"). Implementation is ad-hoc (if/else rules or a manual list) and not scalable, but it provides high-confidence labels for tricky examples. This method is based on expert empirical knowledge (1) and helps correct database or rule mismatches. It is time-consuming, so it is used sparingly for ambiguous formulas.

**4. Probabilistic labeling (3).** For remaining uncertain formulas, one can assign a probabilistic confidence (e.g. via logistic regression on features or label smoothing). For instance, if PubChem shows 80% of similar-composition formulas are stable, we might label with 0.8. This "soft labeling" can train the network to handle uncertainty. Implementation could involve fitting a separate model or using semi-supervised techniques. Reliability depends on data and assumptions (it is not a true rule or ground truth). It combines empirical data frequency with chemical intuition, so it is flagged (3). It is a more complex, lower-confidence method, used after the above steps.

## 4. Chemical Theory–Driven Rules

**1. Octet/Duplet rule and electron counting (2).** The classic heuristic is that main-group atoms prefer a full valence shell (8 electrons, or 2 for H) [1]. We can codify this by checking if total valence electrons match the sum needed for closed shells. For example, $CO_2$ has 16 valence electrons, which exactly fill the octets of C and both O's, so it obeys the rule. A formula that violates this count likely represents a radical or unstable species. Implementation: compute 8*(number of C, N, O, F...)* + *2*(number of H, He) minus total valence electrons; large discrepancies indicate instability. This rule is easiest to implement among chemical heuristics.

**2. Typical valence bond count (formal charge minimization) – [2].** A related rule is that each atom should make its usual number of bonds (C=4, N=3, O=2, etc.), yielding zero formal charge [7]. We can implement

this by checking if each atom's valence electrons could form that many bonds without leaving charges. For instance, if carbon has 4 valence $e^-$ and could use them in 4 bonds, its formal charge is zero. If a structure requires excess or deficit electrons (e.g. $CH_2$=N=N carries formal charges), it tends to be less stable. Coding this fully requires (potentially) generating Lewis structures, but a simplified feature is "sum of (valence $e^-$ – typical bonds*2)". Small sums (near 0) indicate octet satisfaction. This rule is of moderate difficulty but provides insight: stable Lewis structures minimize formal charge, often placing negative charges on more electronegative atoms [8].

**3. Hybridization and bonding patterns (2).** Atoms have typical hybridization patterns (e.g. C forms 4 single bonds or 2 double bonds, N 3 bonds + lone pair, etc.). We can use these as constraints. For example, $CH_3Cl$ (C with 4 bonds) fits $sp^3$ hybridization and is stable, whereas a hypothetical "CH" (C with only 1 bond) breaks $sp^1$/duplet rules. Implementation: infer the total number of σ and π bonds needed for each element and check feasibility (this often overlaps with octet rules). While not a single easy formula, we can encode simple checks like "C count ×4 + N count ×3 + O count ×2 + H count ×1 should equal number of bonding electrons." Violations suggest instability. This rule is harder to code than pure electron counts, but it leverages common valency patterns.

**4. Special electron-count rules (2).** For non–main-group atoms, other rules apply. The 18-electron rule for transition metals is analogous to the octet: many metal complexes are stable with 18 valence electrons. If the Alphamole project later includes organometallics, this rule could be added (though initial phases likely focus on main-group elements). Likewise, hypervalency (P, S exceeding octet) could be considered via ab-initio valence shell theory. These are more advanced and context-dependent, so we rank them as complex, but they are grounded in chemical physics.

**5. Electronegativity balance (2).** Ionic bonding is favored when atoms have large electronegativity differences (e.g. Na (0.93) and Cl (3.16) form stable NaCl). A heuristic rule is that a bond is stable if the more electronegative atom can accommodate the extra electron density (reflected by a reasonable formal charge). This ties back to formal charge placement (negative on EN atoms). We can include a check: if a bonded pair has $\Delta\chi > \sim 2.0$, the bond is likely ionic and stable only if valences match. While not easy to formalize fully here, recognizing electronegativity differences helps (e.g. C–O is normal covalent, but C–O with C having a positive formal charge is stabilized by O's higher χ). Implementation: this is largely heuristic, but it could inform a feature like "max Δχ".

# 5. Overall Priority Ranking

1. **Basic composition features (counts, N, valence)** – These should be implemented first. The elemental count vector and total valence electrons form the foundation of the model (a neural net cannot learn without numeric inputs). They are trivial to code and immediately partition molecules by obvious cases (e.g. odd–electron radicals vs closed-shell). Early phases (N=2) should start here.

2. **Data preprocessing (parsing and scaling)** – Concurrent with feature coding, set up the data pipeline. Formula parsing and standardization of features must be done before any modeling. It's easy but essential: without it, the neural net cannot train properly. Balanced sampling or weighting also belongs early, since severe class imbalance can destroy initial experiments.

3. **Label sourcing (PubChem "stable" flags)** – Simultaneously or next, implement automatic labeling via PubChem or a known compound list. This provides the ground-truth signal to train on. It is

relatively easy (via a REST/API query) and yields high-confidence labels for many formulas. Early models depend on good labels, so establishing this empirical source is high priority (phase 1 and 2).

4. **Baseline model training** – With basic features and labels in place, train an initial model. This reveals how well composition alone predicts stability. Likely start with a simple MLP. Evaluate accuracy on N=2. Identify obvious errors (e.g. radicals misclassified).

5. **Add electrogenitivity and atomic properties** – Once the baseline stabilizes, introduce next-level features: average electronegativity, $\Delta\chi$, molecular weight, etc. These come from known tables and require only lookups, so implementation is moderate. They enrich the model with chemical intuition (polarity, atom size). Their effect can be assessed incrementally (phase 2 for N=3, N=4).

6. **Chemical rule features and labeling adjustments** – Next, incorporate rule-based components. For example, include a binary feature for "octet satisfied" or penalize predictions inconsistent with octet. Alternatively, adjust labels via the valence rule confidence (semi-supervised correction). This leans on chemical knowledge and can catch mistakes the model misses. These steps are logically after basic ML results, since their implementation (logic code) is more complex.

7. **Complex preprocessing (balancing, augmentation)** – If the dataset remains skewed or small, apply sophisticated remedies: oversampling stable formulas, cross-validation for robust metrics, etc. These improve model fairness but are lower-level fixes compared to core features.

8. **Extended chemistry rules (hybridization, 18-electron)** – Finally, consider any domain-specific heuristics relevant to larger N or special elements. For example, if expanding beyond N=4 or including transition metals, integrate those rules. These are lower priority until the model shows deficiency on those fronts.

Each priority reflects a trade-off of ease vs impact: we start with what is easiest and most generally useful (composition, parsing) and move toward more nuanced chemical insights. This phased approach ensures early success (e.g. good accuracy on diatomics using counts alone) and progressively meaningful gains (adding electronegativity and valence constraints in later phases).

**Sources:** We draw on known ML best practices (feature scaling [5], one-hot encoding [4], imbalance handling [6]) and chemical knowledge (valence and octet rules [1] [7], electronegativity trends [2]) in designing this pipeline.

---


[1] Octet rule - Wikipedia
https://en.wikipedia.org/wiki/Octet_rule

[2] Periodic Trends - Chemistry LibreTexts
https://chem.libretexts.org/Bookshelves/Inorganic_Chemistry/Supplemental_Modules_and_Websites_(Inorganic_Chemistry)/Descriptive_Chemistry/Periodic_Trends_of_Elemental_Properties/Periodic_Trends

[3] Atomistic Line Graph Neural Network for improved materials property predictions | npj Computational Materials
https://www.nature.com/articles/s41524-021-00650-1

4 One Hot Encoding in Machine Learning | GeeksforGeeks
https://www.geeksforgeeks.org/ml-one-hot-encoding/

5 6.3. Preprocessing data — scikit-learn 1.6.1 documentation
https://scikit-learn.org/stable/modules/preprocessing.html

6 Random Oversampling and Undersampling for Imbalanced Classification - MachineLearningMastery.com
https://www.machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

7 8 8.5: Lewis Structures - Chemistry LibreTexts
https://chem.libretexts.org/Bookshelves/General_Chemistry/Book%3A_General_Chemistry%3A_Principles_Patterns_and_Applications_(Averill)/08%3A_Ionic_versus_Covalent_Bonding/8.05%3A_Lewis_Structures