Process modeling and high-throughput thermochemical calculations using ChemApp for Python

Ö. K. Büyükuslu, F. Tang, M. to Baben, S. Petersen

GTT-Technologies, 52134 Herzogenrath, Germany, Email: sp@gtt-technologies.de

Keywords: thermochemistry, materials informatics, high-throughput calculations, ChemApp

ABSTRACT

GTT-Technologies' ChemApp for Python was developed to provide a powerful, easy to use interface to ChemApp for a programming language highly popular with scientists and engineers. It is used, for instance, by GTT to develop program modules such as the CALPHAD Optimizer for the FactSage software, by customers to move from interactive FactSage calculations to perform versatile scripting with Python, and by GTT and its partners in research projects in the area of materials informatics.

Computational thermochemistry is fundamental for advancing sustainable metallurgy and creating new alloy compositions for engineering applications. Materials informatics involves handling vast amounts of data and complex workflows.

GTT's approach uses ChemApp for Python and the FactSage thermodynamic databases to design recyclable alloys from the start, incorporating a higher percentage of scraps while aiming to simplify the workflows to simulate material design steps. Challenges arise due to recycling scraps, introducing more elements for consideration. CALPHAD-based databases accurately cover materials from primary metallurgy, but additional data for minority and critical elements is crucial for precise computational modelling.

GTT combines machine learning-based ab-initio databases with traditional CALPHAD databases to cover the complete chemical space with appropriate accuracy. The design of a hardfacing alloy through a high-throughput materials informatics approach is used as a demonstrator of the current possibilities.

INTRODUCTION

ChemApp is a thermochemical software library which enables the user to perform thermochemical calculations across a wide spectrum of applications by providing a programmable interface to perform complex equilibrium calculation techniques for multicomponent, multiphase chemical systems. It is based on Gunnar Eriksson's SOLGASMIX code, which was further developed into ChemSage (Eriksson and Hack, 1990), and became a widely used program for the calculation of complex chemical equilibria.

Since 1996, ChemApp has been available as a product and is not only used as a module for custom program development in research and industry, but also as an add-on to third-party software (Petersen and Hack, 2007). The wide range of application areas is supported by the amount of thermochemical data available for ChemApp. In particular, all thermochemical data accessible through FactSage (Bale et al, 2016) can be used with ChemApp by exporting a subset of the data for a particular chemical system from one or more databases to a data-file.

Initially, ChemApp was used primarily with programming languages such as FORTRAN, C/C++, Basic, and Object Pascal/Delphi. In the last years however, Python became more and more a programming language of choice for scientists and engineers, especially for scripting and prototyping tasks. It was thus decided to develop an interface to ChemApp in the form of a Python package to make ChemApp accessible to a larger group of users and applications.

CHEMAPP FOR PYTHON

Design goals

ChemApp for Python provides several augmentations and additional components that allow for ease of use and increased productivity. Special care was taken to help new users getting started with the packages, without taking away from the accessibility of the raw API to the calculation core, which, due to the potential intricate nature of the calculations to perform, is also frequently used. Therefore, the package contains a *basic* module, which provides a direct link to the ChemApp subroutines, very similar to the well-established C/C++ and Fortran interfaces.

An addition to ChemApp for Python is the *friendly* module, which is exclusive to ChemApp for Python. It strives to simplify the process of setting up and running calculations, making it accessible to users with all levels of expertise.

Furthermore, a set of helper functions and classes are part of ChemApp for Python that allow for collaboration and combination with various typical components of the Python ecosystem, such as pymatgen (Ong et al, 2013), for instance by providing a compatible class for managing and manipulating chemical compositions.

Additionally, a technical necessity shapes a number of decisions regarding the implementation of ChemApp for Python. Since the ChemApp calculation kernel is very strictly procedural, it is unfeasible to keep state in a manner like how Python usually does, by reference-counting of objects and shallow copying. Therefore, an abstraction of result objects is introduced, which bridges the conceptual differences sufficiently well.

For most applications, including the high-throughput calculations introduced in this study, Python is not used for its performance, but rather for its flexibility of use. Nonetheless, performance is a critical feature of every software, and in a way a matter of sustainability, too. Therefore, ChemApp for Python strives to be as thin of a layer as possible, which, especially given the discrepancies in software architecture, is a challenging task.

Implementation details

The underlying ChemApp library is designed with the so-called TQ-Interface, for which all separate dimensions of a thermochemical system loaded from a data-file have to be addressed using an internal index. These indexes exist for phases, system components (which in most cases are the elements of the thermochemical system), but also phase constituents, which are generally phase model-specific countable entities such as species in a gas phase, or certain stoichiometric compositions for which individual modelling data exists.

The problem space is typically defined by several degrees of freedom that are determined by the dimensionality of the thermochemical system, e.g. the number of linearly independent system components, and the number of boundary conditions determined by the Gibbs phase rule.

Despite having its advantages, the setup of calculations using indexes can be improved to increase user-friendliness. ChemApp for Python thus implements an interface that allows for the use of names of phases, system components, and phase constituents, thus increasing maintainability of the code significantly. Furthermore, some of the TQ-Interface functions use string literals. These have all been encapsulated into Python enumeration types, which allow for their type-checking by modern 'linters', which help to prevent coding errors by highlighting semantic and stylistic problems in the source code. As of the management of errors, failure modes of the ChemApp calculations are properly encapsulated into Python exceptions to allow for typical pythonic try-except idioms.

Despite these improvements, the procedural architecture of the ChemApp library and its inability to communicate the internal calculation state continuously are a big problem when trying to produce recreatable results. This means that the calculation path to a solution may be different depending on previous results and calculations, which can be helpful if calculations are similar and therefore the 'proximity' can be used as an advantage. However, it can be also detrimental to performance if, for instance, a randomized input parameter space is employed, for which the final conditions of a calculation are very dissimilar. It is computationally impossible to infer which of those cases is more likely to occur for a given calculational sequence without a deep understanding of the specific calculations.

ChemApp for Python in its entirety is written in Cython (Behnel et al, 2011), resulting in an elegant and efficient wrapping of the ChemApp library, as well as a statically compiled, performance-enhanced, fully C-compiled Python package.

In the design of ChemApp for Python, the developers tried to achieve a balance between the addition of ‘verbose’ commands that are close to natural language, and descriptions which are brief and clear. One example is the group of user-friendly class functions to retrieve specific results (Figure 1).

```
# set the amount of O2 in the gas phase of stream #2 to 'A'
casc.set_IA_pc("#2", "gas_ideal", "O2", A)

# calculate without printing results
casc.calculate_eq(print_results=False)

# get the amount for each phase constituent in the gas (as list)
amount_of_pcs_in_gas = casc.get_eq_A_pcs_in_ph("GAS")

# set the status of SLIQ phase as ENTERED
cats.set_status_ph("SLIQ#1", Status.ENTERED)
```

Figure 1. Example calls for functions that ChemApp for Python provides. Notice that a few abbreviations are being used. These are used throughout the package, making the developer experience fully consistent. The ‘caec’ and ‘casc’ objects are abbreviated class names, with their full names being ChemApp.EquilibriumCalculation and ChemApp.StreamCalculation, respectively.

Having a consistent and concise, but at the same time clear way to retrieve result values was a main goal of the development.

Within the scope of this type of application, the efficiency of high-throughput computations is significantly enhanced by adopting a 'fail early' approach, which can be quickly implemented in Python. The ‘brute force’ of using nested iterations on certain compositions has to be organized in a way so that the most significant criteria can be evaluated first, and in case of failing validation, the nested iterations can be skipped. In Figure 2, an abbreviated example approach highlights how this approach removes a (potentially large) chunk of compositions that are known to fail the criteria later.

```
def critical_cost_reached(B, Mo, Ti):
    # this assumes a global dictionary 'cost' exists with the cost of each
    # element. The value 30 is arbitrarily chosen
    return cost[B] + cost[Mo] + cost[Ti] >= 30

for B in B_range:
    for Mo in Mo_range:
        for Ti in Ti_range:
            # when cost for B, Mo, Ti reaches a certain threshold, skip
            # the rest of the loop because it's going to be too expensive
            if critical_cost_reached(B, Mo, Ti):
                # skip the rest of the loop
                continue
            # otherwise, continue to sample the rest alloying elements
            for Cr in Cr_range:
                ...
```

Figure 2. Slightly modified example of an early failing iteration of the calculation scheme, where injecting an additional condition early to omit certain iterations increases overall performance.

ChemApp calculations can produce vast amounts of reasonably informative data, but it is crucially important to be able to navigate these results. Therefore, a balance must be found between stored and discarded results. As some of the results aren't simple to recover without redoing a calculation,

a satisfactory amount of data needs to be stored to validate against all criteria. Typically, when using ChemApp for Python, a Python object that collects all obtainable results can be generated after each calculation. However, depending on the size of the thermochemical system (e.g. number of system components and phases), this object can reach sizes of a few megabytes per calculation relatively easily, which affects performance and agility of any further analysis. If enough consideration of later-applied criteria can be carried out beforehand, it may be more reasonable to only fetch those required results and omit the generation of the full object. The caveat for that strategy is obviously a loss of possible further evaluations that would require more information. As many strategies of data warehousing in industrial processes prioritize to keep as much data as possible, simply storing all result objects may be more feasible, and ChemApp for Python provides routines to serialize into common database compatible formats to address this need.

LEVERAGING THE PYTHON ECOSYSTEM

A major advantage of using ChemApp for Python for process modeling is the strength of the Python ecosystem that allows for easy data transformations. An example of such a strong incorporation of established data pipelines and the ease of inputting and outputting into adequate formats and plots is illustrated for a simple LD converter process, which can be modelled in a short amount of time. The code for this example can be found online (ChemApp Examples Repository, 2024).

In this example, a specific aspect of the LD converter process of a hot iron melt is illustrated, namely the removal of carbon from the liquid phase by oxygen blowing. In real processes, the composition of the liquid metal varies, with the appropriate amount of oxygen and the reaction enthalpy being the technically interesting results. In Figure 3, an easy way to interact with an externally provided table of input values is illustratively shown to indicate the low threshold that is needed to combine external data sources with ChemApp for Python. Noteworthy is that the *friendly* interface performs all necessary conversions and datatype management issues internally.

```python
import pandas as pd

# content of the hotmetal.csv file:
# > ID,mass,T,Fe,C,Si,Mn,P,S
# > 0,97.09,1306.1,94.88,4.01,0.54,0.43,0.109,0.019
# > 1,99.12,1361.6,94.71,4.28,0.51,0.39,0.096,0.02
#  ...

# read the input data from a CSV file
input_data = pd.read_csv("hotmetal.csv").set_index("ID")

for ID, composition in input_data.iterrows():
    total_mass, T, Fe, C, Si, Mn, P, S = composition
    casc.set_IA_pc("#1", "Fe_bcc(s)", "Fe_bcc(s)", Fe)
    casc.set_IA_pc("#1", "C_Graphite(s)", "C_Graphite(s)", C)
    casc.set_IA_pc("#1", "Si_solid(s)", "Si_solid(s)", Si)
```

Figure 3. A simple way to include external data sources into a Python script, and subsequently into a ChemApp calculation. The code is abbreviated and does not show a fully working example, but simply highlights the few lines of code it takes to import values from a csv file for use as parameters to ChemApp calls.

All calculations, but especially large-dimension calculations benefit a lot from easy exploratory plotting and interaction with reduced data. In Figure 4, a brief example is shown that again highlights the construction of specific data views from a CalculationResultObject and its properties.

```
# create a DataFrame with the columns: A, Amount Fe-liq, Carbon content,
# wt% C, dH
carbon_content_in_Fe_liq = pd.DataFrame(
    columns=["A", "Amount Fe-liq", "Carbon content", "wt% C",
"dH"]).set_index("A")
for ID in input_data.index:
   # iterate over the dictionary of calculated result objects
    for amount_O2, calc_res in converter_results[ID].items():
        # the total phase amount
        amount_feliq = calc_res.phs["Fe-liq"].A
        # the total enthalpy change
        dH = calc_res.dH
        # the phase constituent amount
        amount_c = calc_res.phs["Fe-liq"].scs["C"].A
        # calculate carbon content
        wp_c = amount_c / amount_feliq * 100
```

Figure 4. Using pandas' DataFrames, a table is filled with data specific to a query of the stored results is generated by iterating over the results objects and excerpting the respective values from the objects. The use of pandas' DataFrames allows for very convenient use of the data in subsequent processing pipelines, as they are a de facto standard for data analysis in Python.

HIGH-THROUGHPUT SCREENING USING CHEMAPP FOR PYTHON: A WORKFLOW FOR THERMODYNAMICALLY-INFORMED DECISION MAKING

High-throughput screening (HTS) is a powerful approach in materials science that facilitates the rapid evaluation of numerous material candidates under varying process conditions. Using ChemApp for Python, researchers can systematically explore a wide range of materials and process parameters to make informed decisions. The decision-making process follows a number of steps to narrow down the initial compositional space to a reduced number of candidates for a more thorough examination. This approach is demonstrated below using a simplified example of a high-temperature alloy selection process in the Fe-Cr-Co metallic system.

Step 1: Collect Data

The first step in HTS involves gathering comprehensive data on the materials of interest and their associated process conditions. This data collection includes identifying the state-of-the-art materials and the specific conditions under which these materials will be processed. Key factors to consider are the chemical composition, phase stability, and potential reactions of the materials under certain conditions.

> **Example:** The alloy system Fe-Cr-Co exhibits the sigma phase over a large composition range. The sigma phase is known to cause embrittlement at high temperatures and is not desired.

Step 2: Analyse

Once the data is collected, the next step is to translate the requirements for the material and the process into computable criteria. This involves defining the thermodynamic properties and constraints that are critical for the application. For instance, desired properties such as phase stability, reaction enthalpies, and Gibbs free energy changes are converted into specific criteria that can be analysed computationally.

> **Example:** The compositional space for the Fe-Cr-Co system is generated based on the literature/plant data as shown in Table 1. The equilibrium calculations will be performed for a

temperature of 800°C at ambient pressure. The activity of the sigma phase will be stored in a results database for each composition to assess the stability of the embrittling phase.

Table 1. Composition space for the Fe-Cr-Co system. “Stepsize” represents the resolution of the calculations and corresponds to the increment of system component amounts from minimum to maximum (in wt.%).

System component	Min (wt.%)	Max (wt.%)	Stepsize (wt.%)
Fe	10	80	balance
Cr	20	80	2
Co	10	70	1

Step 3: Calculate

With the criteria defined, the next step is to perform the thermochemical calculations using ChemApp for Python. This involves setting up and executing a series of equilibrium calculations to generate the necessary thermochemical results. ChemApp for Python allows for the automation of these calculations, enabling the efficient handling of a large number of scenarios. The results from these calculations, including equilibrium compositions, phase distributions, and thermodynamic properties, are stored for further analysis.

Example: ChemApp for Python code snippet. The composition space is generated and fixed conditions are set (1). The incoming amount for each system component is defined (2) and the equilibrium for each composition is calculated (3) within a nested loop. The activities of the sigma phase are stored in a results dataframe (4). In case the equilibrium cannot be calculated, the exception handler is triggered (5).

```
# Set fixed conditions (1)
Co_range = np.arange(10, 71, 1)
Cr_range = np.arange(20, 81, 2)
caec.set_eq_T(800)  # °C
caec.set_eq_P(1)  # bar
for Co in Co_range:
    for Cr in Cr_range:
        Fe = 100 - Co - Cr
        if Fe <= 80 and Fe >= 10:
            # Set chemical formula incoming amounts. (2)
            caec.set_IA_cfs(["Fe", "Co", "Cr"], [Fe, Co, Cr])
            try:
                # Calculate equilibrium (3)
                caec.calculate_eq()
                activity_sigma = caec.get_eq_AC_ph("SIGMA")
                # Assume the results are stored in a
                # dataframe with the composition as index (4)
                results_df.loc[Fe, Co, Cr] = activity_sigma
                print(f"composition {composition_ID} is calculated")
            # Handle error (5)
            except ChemAppError:
                print("Equilibrium cannot be calculated!")
                results_df.loc[Fe, Co, Cr] = "None"
```

Figure 5. High-throughput screening of alloy candidates using ChemApp for Python. The code snippet illustrates the procedure for setting conditions, calculating the equilibrium, and retrieving results.

Step 4: Filter & Select

The final step in the HTS workflow is to filter and select the most promising material candidates based on the predefined criteria. This involves comparing the calculated results against the desired properties and constraints. Candidates that meet the criteria are shortlisted for further experimental validation or a more detailed study. This step ensures that only the most viable materials, which satisfy all necessary thermodynamic requirements, are considered for further examination.

> **Example:** Set a threshold for the activity of the sigma phase. The compositions that pass the criteria are stored in a new dataframe, which is subsequently used for a new set of calculations. Repeat the screening until all computable requirements are met based on the materials design and processing conditions.

Input:

```
threshold = 0.5
results_df_filtered = results_df[results_df["activity_sigma"] < threshold]
print(results_df_filtered)
print(
    f"""Out of {results_df.shape[0]}, {results_df_filtered.shape[0]}
    compositions have a sigma phase activity of less than {threshold}."""
)
```

Figure 6. Postprocessing of equilibrium calculation results. The code snippet illustrates how the equilibrium calculation results are filtered to identify compositions where the sigma phase has an activity of less than 0.5. It then prints the filtered results and reports the number of compositions meeting the threshold criteria.

Output:

```
                        activity_sigma
Fe_wt% Co_wt% Cr_wt%
21     59     20             0.497369
20     60     20             0.478664
19     61     20             0.461042
18     62     20             0.444460
17     63     20             0.428878
16     64     20             0.414260
15     65     20             0.400567
14     66     20             0.387766
13     67     20             0.375822
12     68     20             0.364705
11     69     20             0.354385
10     70     20             0.344834

Out of 961, 12 compositions have a sigma phase activity of less than 0.5.
```

Figure 7. Output of results generated by the code snippet shown in the Figure 6.

By following the steps outlined above, researchers can make thermodynamically informed decisions that enhance the efficiency and effectiveness of material selection and process optimization. Screening can be repeated to further refine the composition space of alloy candidates.

CASE SCENARIO: HARDFACING ALLOY DESIGN USING THERMOCHEMICAL SCREENING

In this example we introduce the design of a hardfacing alloy using a high-throughput “materials informatics” approach. The objective is to identify successful compositions for an iron-based hardfacing alloy for mining and agricultural applications, which is also cheaper than a patented benchmark material. The project started with a sample space of 765.000 compositions based on a range of the benchmark alloy’s reference composition space that involves 12 elements. As a result of the multi-step HTS process, 168 promising compositions were obtained.

The foundational step in the high-throughput approach involves the selection and integration of compatible thermochemical data within the chemical system of interest. FactSage provides a variety of databases which are particularly useful for various applications such as materials design, process metallurgy, and energy conversion. By combining CALPHAD (Calculation of Phase Diagrams) and ab-initio based thermochemical data for use with ChemApp calculations, users can address complex scenarios in material science, such as identifying critical elements in steel metallurgy, which require a comprehensive and accurate materials property database.

Following the selection of the appropriate thermochemical data, specifications must be clearly defined based on the intended application of the material. This corresponds to step 1 (see above), where data is collected for the calculations. Specifications can be material-, process- or cost-related based on the application case. A crucial step is the transformation of process specifications to thermodynamically computable criteria in order to generate a high-throughput calculation workflow (i.e. analyse the data and ensure that the specifications are met). For instance, the design of hardfacing alloys necessitates specific processes and properties such as austenitic solidification, martensitic transformation, and a narrow melting range. The melt range criterion is defined as the difference between the formation temperature of a first hard phase to solidify, and the liquidus temperature of a matrix phase. It is a fingerprint of the tendency for hot cracking during solidification. Additionally, it is vital to avoid conditions that result in the formation of phases such as carbides, borides, and intermetallics that can embrittle the alloy. Specific conditions also include physical properties such as low density and cost-effectiveness, which can be estimated using databases such as GTT’s ab-initio Materials Project Database (aiMP). These specifications can serve as computable criteria for the subsequent material screening process.

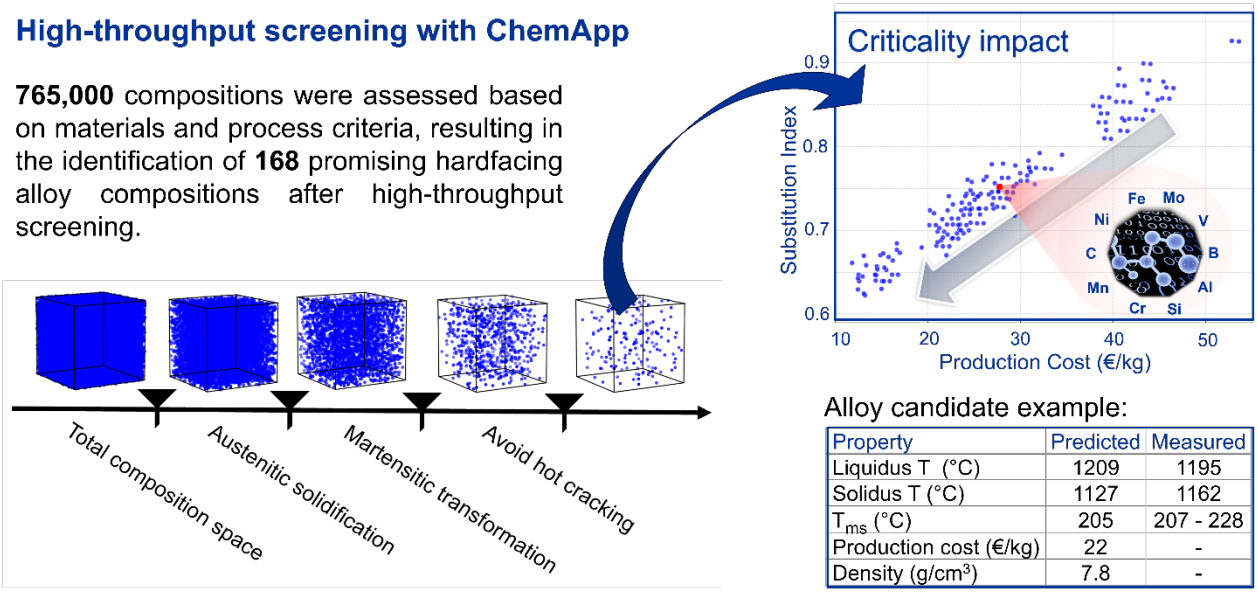
The next phase involves navigating the chemical space to identify potential candidates that meet the defined specifications. ChemApp offers robust capabilities for exploring and manipulating chemical compositions. This allows researchers to rapidly scan through thousands of potential compounds to pinpoint those that align with the desired material properties. In the process of generating the compositional space for the design of a hardfacing alloy, it is initially defined to consist of 10 metals plus boron (B) and carbon (C). The composition range for each element is established with iron serving as the balancing element. The maximum and minimum values (in mol.%) are set for each component. Iron is set to vary from 45 to 70 mol%, while additives like molybdenum, vanadium, and nickel vary from 0 to 20 mol%, and carbon and boron from 5 to 15 mol%. The step sizes for each element are set to values between 2.5 and 5 mol%. This structured approach allows for the systematic exploration and analysis of the defined chemical space.

Once the compositional space is generated, it is possible to calculate and perform the equilibrium calculations. The initial screening of 765,000 compositions involves two equilibrium calculations at temperatures of 500°C and 1000°C. This process aims to identify compositions that predict stability for the FCC (face-centred cubic) phase at high temperature (1000°C) and the BCC (body-centred cubic) phase at low temperature (500°C), aligning with the benchmark alloy and its FCC-BCC transformation criteria. Additionally, the screening checks for the predicted absence of the graphite phase, which is crucial for the desired material properties. Once the calculation results are stored in a dedicated database, the compositions can be subsequently refined, corresponding to the “Filter & Select” step of the high-throughput approach. This efficiently narrows down the number of potential candidates by assessing their relevant phase stabilities under the specified thermal conditions.

After filtering out 92% of compositions as a result of the initial screening, the focus shifts to more computationally intensive calculations, now manageable due to the reduced dataset size. The melt range of hard precipitates is determined through a two-step process that starts with the calculation of the liquidus temperature (Tliq) of the matrix phase. This temperature is critical for identifying the point at which the material begins to solidify. Following this, an equilibrium calculation is conducted at 250°C lower than Tliq to assess the stability of the matrix phase at reduced temperatures. The martensite start temperature (Tms) model, which predicts the temperature at which martensite (a harder, more brittle phase of steel) begins to form, is applied to successful candidates that pass the melt range criteria (van Bohemen and Morsdorf, 2017). This prediction is based on how the free energy of the material changes with temperature between two different Fe-matrix phases (FCC and BCC) at 1000°C.

Estimated Tms values agree well with experimental findings of our project partners within a range of ±50°C. The aiMP database is employed to calculate the bulk density of each alloy composition. The cost of each composition is calculated by the weighted average of phases and corresponding densities. Approximately 300 phases are identified for each composition and categorized into four groups: (1) Fe-matrix phase, (2) carbides & borides that cause strengthening, (3) carbides & borides that cause embrittlement, and (4) intermetallics. Consequently, any compositions that include phases leading to embrittlement (i.e. which belong to group 3 and 4) are filtered out. The remaining compositions are taken as the successful candidates that pass the screening criteria. Furthermore, the solidus and liquidus temperatures of the alloy are reported for each successful composition, as these temperatures define the range over which the alloy remains partially liquid and are thus crucial for processing techniques in additive manufacturing.

In the final screening step, alloy compositions which pass the thermochemical screening criteria are evaluated based on the criticality of the alloying elements. Criticality in the context of alloying elements refers to the assessment of risks associated with their supply, which includes factors such as scarcity, geopolitical risks, and environmental implications of their extraction and processing. As shown in Figure 8, the “Substitution Index” (SI) is used as a metric for criticality, where the arrow indicates the tendency towards a lower substitution index that is the goal with respect to the alloy composition, as well as a lower production cost.



Alloy candidate example:

Property	Predicted	Measured
Liquidus T (°C)	1209	1195
Solidus T (°C)	1127	1162
T_{ms} (°C)	205	207 - 228
Production cost (€/kg)	22	-
Density (g/cm^3)	7.8	-

Figure 8. *High-throughput screening of alloy candidates using ChemApp for Python. The figure demonstrates the process of identifying suitable alloys through high-throughput screening. The methodology encompasses evaluation within the total compositional space, austenitic solidification, martensitic transformation, and avoidance of hot cracking (e.g. melt range criteria). The criticality impact plot (top right) highlights the substitution index versus production cost (€/kg). The arrow indicates the intended tendency towards a lower substitution index and lower productions costs for the alloy composition. Predicted results and measured properties by subsequent processing of a single alloy candidate are given in the inserted table (bottom right). Liquidus, solidus and martensite start temperatures are compared and found to be in good agreement.*

An average substitution index is calculated based on the aforementioned criteria for each composition and is found to be in good agreement with the low production costs associated with using fewer alloying elements.

The thermochemical screening methodology used in this work has proven to be effective in identifying potential alloy candidates for industrial applications, taking into account process specifications, production costs, as well as environmental and supply chain of issues of critical alloying elements. Successful alloy candidates already underwent experimental analysis by our project partners. It was found that the measured findings align well with the predicted results, providing corroborating evidence for the existence of novel alloy compositions that are more sustainable and cheaper than the benchmark alloy while maintaining structural performance.

CONCLUSIONS

In the rapidly evolving field of material science, the high-throughput materials informatics approach has emerged as a transformative methodology, significantly accelerating the discovery and optimization of novel materials. With the automatic generation of ChemApp for Python scripts from FactSage, this approach is now easily accessible to every FactSage user, be it in an academic or industrial research context. The potential is demonstrated by screening millions of compositions for interesting properties as hardfacing alloys. The properties include application-related properties such as high hardness by formation of martensite and pre-defined volume fraction of carbides and borides, as well as properties related to processing, such as low liquidus temperature and low tendency for hot cracking during solidification. Besides the direct screening for application- or processing-related properties, the methodology can also be used to generate high-quality, homogeneous input data for machine learning models.

REFERENCES

Bale, C W, Bélisle, E, Chartrand, P, Decterov, S A, Eriksson, G, Gheribi, A E, Hack, K, Jung, I H, Kang, Y B, Melançon, J, Pelton, A D, Petersen, S, Robelin, C, Sangster, J, Spencer, P, Van Ende, M-A, 2016. FactSage Thermochemical Software and Databases 2010 – 2016. Calphad, 54:35-53. https://doi.org/10.1016/j.calphad.2016.05.002

Eriksson, G, and Hack, K, 1990. ChemSage - A computer program for the calculation of complex chemical equilibria, Metall Trans B, 21:1013-1023. https://doi.org/10.1007/BF02670272

Ong, S P, Richards, W D, Jain, A, Hautier, G, Kocher, M, Cholia, S, Gunter, D, Chevrier, V L, Persson, K A, Ceder, G, 2013. Python Materials Genomics (pymatgen): A Robust, Open-Source Python Library for Materials Analysis, Computational Materials Science, 68:314-319. https://doi.org/10.1016/j.commatsci.2012.10.028

Petersen, S, and Hack, K, 2007. The thermochemistry library ChemApp and its applications. International Journal of Materials Research, 98(10):935-945. https://doi.org/10.3139/146.101551

van Bohemen, S M C, and Morsdorf, L, 2017. Predicting the Ms temperature of steels with a thermodynamic based model including the effect of the prior austenite grain size, Acta Materialia, 125:401-415. https://doi.org/10.1016/j.actamat.2016.12.029

Behnel, S, Bradshaw, R, Citro, C, Dalcin, L, Seljebotn, D S, Smith, K, 2011. Cython: The best of both worlds. Computing in Science & Engineering, 13(2), pp.31–39. http://dx.doi.org/10.1109/MCSE.2010.118

ChemApp Examples Repository (2024) *GitHub*. Available at: https://github.com/GTT-Technologies/ChemApp-Examples/ .