Text-to-Flowsheet: An LLM-Assisted Pipeline for Expert-Level Digitization and Automated Simulation of Chemical Processes

- Supplementary Information -

Jan-Frederic Laub,[†,‡] Luca Bosetti,[†] and André Bardow[*,†,‡]

[†]*Energy and Process Systems Engineering, ETH Zurich, Switzerland*

[‡]*NCCR Catalysis, Switzerland*

E-mail: abardow@ethz.ch

This supplementary information contains

- the table of considered unit operations,
- a complete description of the topology generation pipeline,
- a complete description of the chemical augmentation pipeline,
- a complete discussion of the flowsheet similarity metrics,
- a comparison to a "single-prompt" inference mode,
- a comparison between different LLMs, including a much smaller LLM,
- details on the permutation test between expert- and LLM-generated flowsheets,
- the complete distribution of expert similarities,
- examples of expert flowsheet drawings,
- details on the test set processes,
- the objective functions for black-box unit optimization,
- an example of the multi-objective black-box unit optimization,
- the full process descriptions for example 1, 2, and 4, and
- all LLM prompts for "text2flowsheet".

SI 1 Table of Unit Operations and Aspen Models

Table SI 1: List of considered unit operations, their abbreviations, number of incoming and outgoing streams, and preferred model in Aspen Plus (see also Figure 4 of the manuscript). Adapted from Vogel *et al.* (2023).[1]

Unit	**Abbreviation**	**In / Out Degree**	**Preferred Aspen Plus Model**
Absorption	Abs	2/2	Radfrac
Centrifuge	Centr	1/2	Sep
Compressor	Comp	1/1	Compr
Condenser	Cond	1/1	Heater
Cooler	Cool	1/1	Heater
Cyclone	Cycl	1/2	Sep
Distillation (incl. reboiler and condenser)	Dist	1/2	Radfrac
Drying	Dry	1/2	Sep
Evaporator	Evap	1/1	Heat
Expander	Expand	1/1	Valve
Extraction	Extr	2/2	Extract
Extractive Distillation	exDist	1/2	Radfrac
Flash	flash	1/2	Flash2
Gas filtration	Gfil	1/2	Sep
Heater	Heat	1/1	Heater
Heat exchanger	Hex	1/1	HeatX
Hydrocyclone	Hcycl	1/2	Sep
Liquid filtration	Lfil	1/2	Sep
Mixing	Mix	$\geq$2/1	Mixer
Product (Outlet)	Prod	1/0	MATERIAL
Pump	pp	1/1	Pump
Raw Material (Inlet)	Raw	0/1	MATERIAL
Reactor	r	1/1	RGibbs
Scrubbing	Scrub	2/2	Extract
Separation (no further sub-specification)	Sep	1/2	Sep
Settling	Set	1/2	Decanter
Splitting	Splt	1/2	FSplit
Stripping	Strip	$\geq$1/2	Radfrac
Storage	Tank	1/0	MATERIAL
Unknown unit	X	1/1	MULT (dummy)

SI 2 Complete description of the topology generation pipeline

Accompanying Figure 2 of the main manuscript:

First, the LLM is tasked with extracting all unit operations explicitly mentioned, or sufficiently implied, in the process description (Figure 2, a1). Additionally, the LLM extracts all raw-material streams crossing the system boundary and all product, co-product, and waste streams leaving the system. The resulting pool of constitutive unit operations and boundary streams serves as the basis for the units' topological arrangement, i.e., the material flow through the production steps. For each extracted unit, the LLM determines its unit operation type, thereby configuring the desired number and types of the unit's outlet streams (e.g., top/bottom or vapor/liquid). The set of unit types is harmonized with the OntoCAPE process engineering ontology[2] through the SFILES2 framework,[1] as shown in Table SI 1.

Figure 2 shows the overall processing step sequence. The core step in composing a process's topology is a depth-first, iterative graph construction in which the LLM determines the placement of a subsequent node at each open stream (Figure 2, a2). The flowsheet graph is initialized with all raw material streams (defined as nodes) extracted in the previous step (Figure 2, a2a). Then, the LLM is queried to determine the next unit in the process sequence from the candidate pool of unit operations and product streams (Figure 2, a2b). The preliminary topology is complete once depth-first graph construction is exhausted, i.e., every node has a successor or is itself a product stream.

In the next step, the LLM re-examines any leftover unit operations and product streams from the initial candidate pool (Figure 2, a3). The LLM checks whether any leftover unit can be reasonably integrated at any point in the preliminary topology, drawing again on the process description, and discards it if not.

The preliminary topology undergoes checks for logical consistency with process design conventions. For example, any unit operation that should have an outgoing stream but has no following unit or product stream associated with that outlet is assigned a stream leaving the system boundary (Figure 2, a4). For each newly added outlet, the LLM is tasked with determining a possible role of that stream within the process description. Similarly, any unit operation that should but does not yet receive a solvent, entrainer, or similar is equipped with a corresponding inlet, and the LLM again characterizes this stream.

After undergoing further checks for logical consistency with the text and process design conventions, the topology graph closely reflects the structural information in the process description while meeting the basic topological requirements of a process flowsheet. From this point on, the topology is not further manipulated by the LLM and serves as the basis for subsequent information extraction steps that augment it with chemical and operational data.

SI 3 Complete description of the chemical augmentation pipeline

After completing the topology pipeline described in the previous section, the process graph describes the material flow through the process and contains the type of each unit operation. Usually, process descriptions include additional information, such as the names of produced and consumed chemicals, stream compositions, temperatures, and pressures. This data is now extracted from the text by the LLM and assigned to the relevant unit nodes in the graph representation.

First, the chemicals contained in each raw material, product, co-product, and waste stream are extracted from the text. The extracted chemical designations are automatically canonicalized to their IUPAC names using PubChem[3] and to canonical SMILES using RDKit.[4]

Next, the chemicals involved in each reactor unit are extracted and classified. Starting from a reactor unit, all upstream raw materials and all downstream products are gathered. To ensure that no trace compounds or catalysts are overlooked, the LLM is also tasked with extracting all relevant chemical compounds from the reactor description itself. Then, the LLM classifies each chemical associated with the reaction as a reactant, product, byproduct, catalyst, or solvent. Additionally, the main product of the reaction is flagged, which can be the overall product of the process or an intermediate. If available in the text, the LLM also adds the reactor temperature and pressure as attributes of the reactor node. Similarly, relevant operational conditions are recorded for all other units.

If stoichiometries for the process's reactions are available in the process description or associated information, these attributes are added to the reactor nodes. If not, a possible stoichiometry is computed for the extracted chemicals by minimizing the Gibbs free energy of the reaction under the constraint of fulfilled elemental balances. Common reaction side products, such as water and carbon dioxide, are also included in the pool of possible chemicals when computing a mechanistically reasonable stoichiometry. The stoichiometric compositions of the reactant streams are calculated and added as attributes to the raw material streams, serving as potential initialization points for subsequent simulation.

The core element of process data augmentation is the analysis of the process's separation steps. For the eventual simulation, it is critical to determine which component-wise splits are targeted by each separation unit, as specified in the process description. As this information is often absent or only implied in natural language sources, we trace the component splits from products to reactants in the opposite direction to the process's material flow: First, all recycle streams are temporarily torn to convert the flowsheet graph into a tree. Then, a hierarchy of separation nodes is determined based on each separator unit's distance to the

last reactor node. The components of each stream are traced back through the flowsheet level by level, starting from the outlet streams. At each separation node, the outgoing components are aggregated into a list, and the component-wise split at that node is recorded and categorized by outlet stream type (e.g., top/bottom or vapor/liquid). When one of the separator's outlets is not connected to a product node because it is part of the recycle in the original cyclic graph, the LLM is queried to determine the recycled compounds using the process description as context.

Identifying the key components in each stream characterizes the separation tasks. In the case of distillation-like units, the boiling temperature of each compound is computed with FeOs[5] using the experimentally fitted PC-SAFT parameters from Esper *et al.* (2023)[6] if available, or, if not, the predicted PC-SAFT parameters from Winter *et al.* (2025).[7] If the separation pressure or temperature is given, the boiling points are calculated at that condition; otherwise, they are computed at atmospheric conditions. The compounds in the distillate and bottom streams are sorted by relative volatility, and the heavy and light key components are identified. If the determined heavy key has higher volatility than the light key, this check indicates that the LLM incorrectly interpreted the topology of the unit's outgoing streams. Consequently, the positions of the outlet streams are corrected in the graph.

The determined process topology, together with the augmented chemical and operational data, is further processed to enable its automatic translation into Aspen Plus: First, any graph structure is deleted that is not connected to the main process topology, i.e., the largest subgraph that involves a product stream of the processes' main product. Then, a mixing node is inserted before any unit with more than one incoming stream. Finally, paths that start from a raw material node and involve ill-defined chemical species, e.g., due to failed name lookup and standardization, are excluded.

SI 4 Discussion of the flowsheet comparison metrics

In this section, we discuss and select appropriate metrics for flowsheet similarity and a strategy to place them in a domain-specific context through empirically assessed expert flowsheet drawings.

SI 4.1 Topology-invariant similarity

Commenge and Piña-Martinez (2026)[8] define a metric of dissimilarity between two flowsheets based on evolutionary graph manipulations. The metric considers the manipulations of adding nodes (unit operations), deleting nodes, and merging input and output streams at the flowsheet boundary. Given two flowsheet graphs G_1 and G_2, this metric computes the number of manipulation steps required to convert the first graph into a second graph with the same number of units (per type and degree), the same number of streams, and the same number of external inputs and outputs as the second graph. However, this metric does not account for the graphs' inner connectivities. As a result, the metric cannot distinguish between two flowsheets that have the same number of units (per type and degree) and streams, but differ in arrangement.

Nevertheless, the metric could be helpful, as it is rooted in domain knowledge by drawing on common graph manipulations in evolutionary process design (see, for example, Neveux (2018)).[9] Here, we calculate the average dissimilarity $d_{\text{av}}(G_1, G_2) = \frac{d(G_1,G_2)+d(G_2,G_1)}{2}$ based on the instructions in Commenge and Piña-Martinez (2026),[8] normalize it with respect to the total number of nodes (V_1 and V_2) of both graphs, and subtract it from one to arrive at a similarity score bounded between zero and one:

$$S_{\text{evol}}(G_1, G_2) = 1 - \frac{d_{\text{av}}(G_1, G_2)}{|V_1| + |V_2|}. \tag{1}$$

SI 4.2 Topology-aware similarity

To obtain a topology-sensitive yet computationally tractable similarity score, we employ a graph kernel based on the Weisfeiler–Lehman (WL) subtree framework.[10] WL-type graph kernels are typically recommended for sparse graphs,[11] like our flowsheets. The WL kernel measures structural similarity by iteratively refining node labels based on their local neighborhoods. In each WL iteration, every node aggregates the hashed multiset of labels of its adjacent nodes, thereby encoding increasingly larger local substructures. After three refinement steps, the resulting multilevel label histograms serve as feature maps of the two graphs G_1 and G_2. The inner product of the histograms yields the WL kernel value $k_{\mathrm{WL}}(G_1, G_2)$, which grows with the number of identical neighborhood patterns between the graphs.

In our context, the WL kernel seems to be well-suited because it naturally incorporates the unit operation types already encoded as node labels. Thus, two flowsheets that employ similar units but arrange them differently produce distinct WL fingerprints, whereas flowsheets that share both unit types and their connectivity patterns yield more similar feature maps. However, raw kernel scores $k_{\mathrm{WL}}(G_1, G_2)$ are not yet interpretable as similarity values, because they depend on the absolute magnitude of the feature vectors. We therefore normalize the WL kernel to obtain a bounded similarity metric:

$$S_{\mathrm{WL}}(G_1, G_2) = \frac{k_{\mathrm{WL}}(G_1, G_2)}{\sqrt{k_{\mathrm{WL}}(G_1, G_1)\; k_{\mathrm{WL}}(G_2, G_2)}}, \tag{2}$$

which is equivalent to the cosine similarity of the WL-derived feature embeddings. This normalization yields values in $[0, 1]$, with 1 indicating identical WL representations (i.e., matching unit types and topologies under WL refinement) and 0 indicating no shared substructure.

SI 4.3 Flowsheet similarity among expert-level text interpretations

To validate the digitization methodology, we want to compute the similarity metrics between flowsheets automatically generated by the LLM and those generated by experts. While the similarity metrics alone provide a reasonable indication of digitization success, we do not necessarily expect a satisfactory value to be close to 100%, as the process texts usually contain ambiguous information that can be interpreted in different yet equally valid ways. We therefore seek to calculate upper and lower reference points that help to place similarity scores into our domain-specific context.

To account for ambiguities in the text descriptions, a meaningful upper target range of digitization success can be determined by assessing the average similarity between flowsheets drawn by experts for the same process description. For this purpose, we have gathered a total of 101 expert-drawn flowsheets for 30 processes and computed pairwise similarity scores. As expected, the similarity scores rarely exceed 80% (see Figure 4), indicating that experts regularly disagree to some extent on how to represent the given textual information in a flowsheet drawing. The distribution of pairwise expert similarities provides a measure of uncertainty that helps contextualize eventual validations of the LLM-generated flowsheets.

The medians of the two metrics are to some extent correlated, with a Pearson correlation of 0.78, because the WL metric also implicitly accounts for unit-type distributions. However, in many cases, the evolutionary metric yields greater similarity scores by disregarding topological differences among similar units.

Nevertheless, the two metrics provide distinct information for analysis: Figure 3 (manuscript) shows the distributions of pairwise similarities between expert-drawn flowsheets for two exemplary processes from the test set, using both similarity metrics. Process text #5 describes one reactor and four subsequent distillation steps. All experts correctly place one reactor

and four distillation columns in their flowsheets, but interpret the number and destination of the described recycles differently (see supplementary information†). The differing topologies have only a weak, indirect impact on the evolutionary score, through some missing outlet streams, but are clearly reflected in the comparatively lower WL scores and their wider spread. Compare this topological effect on the scores with the flowsheet similarities for process #18: Here, many experts largely agree on the unit types and overall topological structure of the process, as reflected in the evolutionary and WL scores. Expert 2's interpretation of process #18 (see supplementary information†) is the least similar to all other expert drawings, particularly due to the handling of a heat-integrated stream, which affects the WL score's distribution as seen in Figure 3 (manuscript). For these reasons, we employ both metrics for expert correspondence as the upper target ranges for the accuracy of the LLM-assisted digitization pipeline.

The random graphs are generated uniformly from a range of $[N^{\text{exp}}_{\text{min}}, N^{\text{exp}}_{\text{max}}]$ nodes and edge densities within $[\rho^{\text{exp}}_{\text{min}}, \rho^{\text{exp}}_{\text{max}}]$, where N^{exp} represents the number of nodes and ρ^{exp} the edge densities of the process's expert flowsheets. To generate the random, flowsheet-like graphs, first, a random tree is generated, and then edges are appended until the desired edge density is reached, yielding flowsheet-like (weakly connected, directed, sparse) yet random graph structures. The random similarities are considerably lower for the WL score than for the evolutionary metric because random graphs can differ in many more ways when considering graph connectivity in addition to the distribution of unit types.

SI 5 Comparison to "single-prompt" inference mode

We posed the topology generation and chemical augmentation task to the LLM GPT-OSS as a single prompt, rather than the multi-step pipeline presented in the manuscript, under otherwise identical conditions. The used DSPy signature was:

```
class OnePromptFlowsheetGeneration(dspy.Signature):
    """
    You are an expert chemical process engineer. Given a process
   description and its main product, generate a complete, fully annotated
   process flowsheet in a single structured response.

    UNIT IDs: Assign sequential IDs U_1, U_2, U_3, ... to all units.

    UNIT TYPES and their required stream labels (keys of following_units):
      Raw:          {"raw": <next_unit_id>}
      Prod:         {}
      Mix:          {"main": <next>}
      r (Reactor):  {"main": <next>}
      Dist:         {"top": <next>, "bottom": <next>}
      Rect:         {"top": <next>, "bottom": <next>}
      azDist:       {"top": <next>, "bottom": <next>}
      exDist:       {"top": <next>, "bottom": <next>}
      reDist:       {"top": <next>, "bottom": <next>}
      Strip:        {"top (vapor)": <next>, "bottom (liquid)": <next>}
      Abs:          {"vapor": <next>, "liquid": <next>}
      Scrub:        {"vapor": <next>, "liquid": <next>}
      flash:        {"vapor": <next>, "liquid": <next>}
      Extr:         {"raffinate": <next>, "extract": <next>}
      Sep:          {"light": <next>, "heavy": <next>}
      Set:          {"light": <next>, "heavy": <next>}
      Splt:         {"split_1": <next>, "split_2": <next>}
      Crys:         {"liquid": <next>, "solid": <next>}
      Centr:        {"centrate": <next>, "discharge": <next>}
      Cycl:         {"centrate": <next>, "discharge": <next>}
      Hcycl:        {"centrate": <next>, "discharge": <next>}
      Dry:          {"water": <next>, "dry": <next>}
      Gfil:         {"permeate": <next>, "retentate": <next>}
      Lfil:         {"permeate": <next>, "retentate": <next>}
```

```
    Heat/Cool/Hex/Evap/Comp/Cond/pp/Expand/Reb/X: {"main": <next>}

 FIELD RULES per unit type:
 - Raw / Prod:
     chemical_compounds: list the exact chemical name(s) in that feed/product stream.
 - r (Reactor):
     reaction_compounds: classify every chemical involved (reactant, product, byproduct, catalyst, solvent).
     reaction_temperature_K
     reaction_pressure_bar
 - Dist / azDist / exDist / reDist / Rect / Strip:
     distillation_top_pressure_bar, distillation_bottom_pressure_bar
     separation_streams: one entry per output label (e.g., label="top", chemicals=[...])
     light_key
     heavy_key
     azeotrope_exists
     azeotrope_components
 - Abs / Scrub / Extr / Sep / Set / Crys / flash / Centr / Cycl / Lfil / Gfil / Hcycl / Dry:
     separation_streams, light_key, heavy_key

 all_components: the deduplicated list of ALL chemical species appearing anywhere in the flowsheet. Use IUPAC names or standard accepted names.

 Return your answer as a single JSON object:
 {
   "units": [
     {
       "unit_id": "U_n",
       "name": "<descriptive name>",
```

```
        "type": "<one of the types listed above>",
        "description": "<description>",
        "function": "<production|maintenance|unclear>",
        "following_units": {"<stream label>": "<unit_id or null>"},
        "chemical_compounds": ["<chemical>"],
        "reaction_compounds": [
          {
            "name": "<chemical>",
            "role": "<reactant|product|byproduct|catalyst|solvent|
maintenance|N/A>",
            "is_main_product": "<True|False>",
            "is_base_reactant": "<True|False>"
          }
        ],
        "reaction_temperature_K": null,
        "reaction_pressure_bar": null,
        "distillation_top_pressure_bar": null,
        "distillation_bottom_pressure_bar": null,
        "separation_streams": [{"label": "<stream label>", "chemicals":
["<chemical>"]}],
        "light_key": ["<chemical>"],
        "heavy_key": ["<chemical>"],
        "azeotrope_exists": false,
        "azeotrope_components": []
      }
    ],
    "all_components": ["<chemical>"]
  }
  """
  process_description: str = dspy.InputField()
  main_product: str = dspy.InputField()
  flowsheet: str = dspy.OutputField()
```

Figure 1 summarizes the performance of the "single-prompt" setup compared to the multi-step pipeline on the test set of expert drawings. The "single-prompt" setup failed to produce a flowsheet in 22 out of 30 cases. As a consequence, the single-prompt outputs cannot be reliably processed for comparison to expert flowsheets or Aspen Plus simulations. In the eight cases where the LLM does produce a graph topology with a single prompt, is it on average less similar to the expert interpretations than the flowsheets produced by the full pipeline.

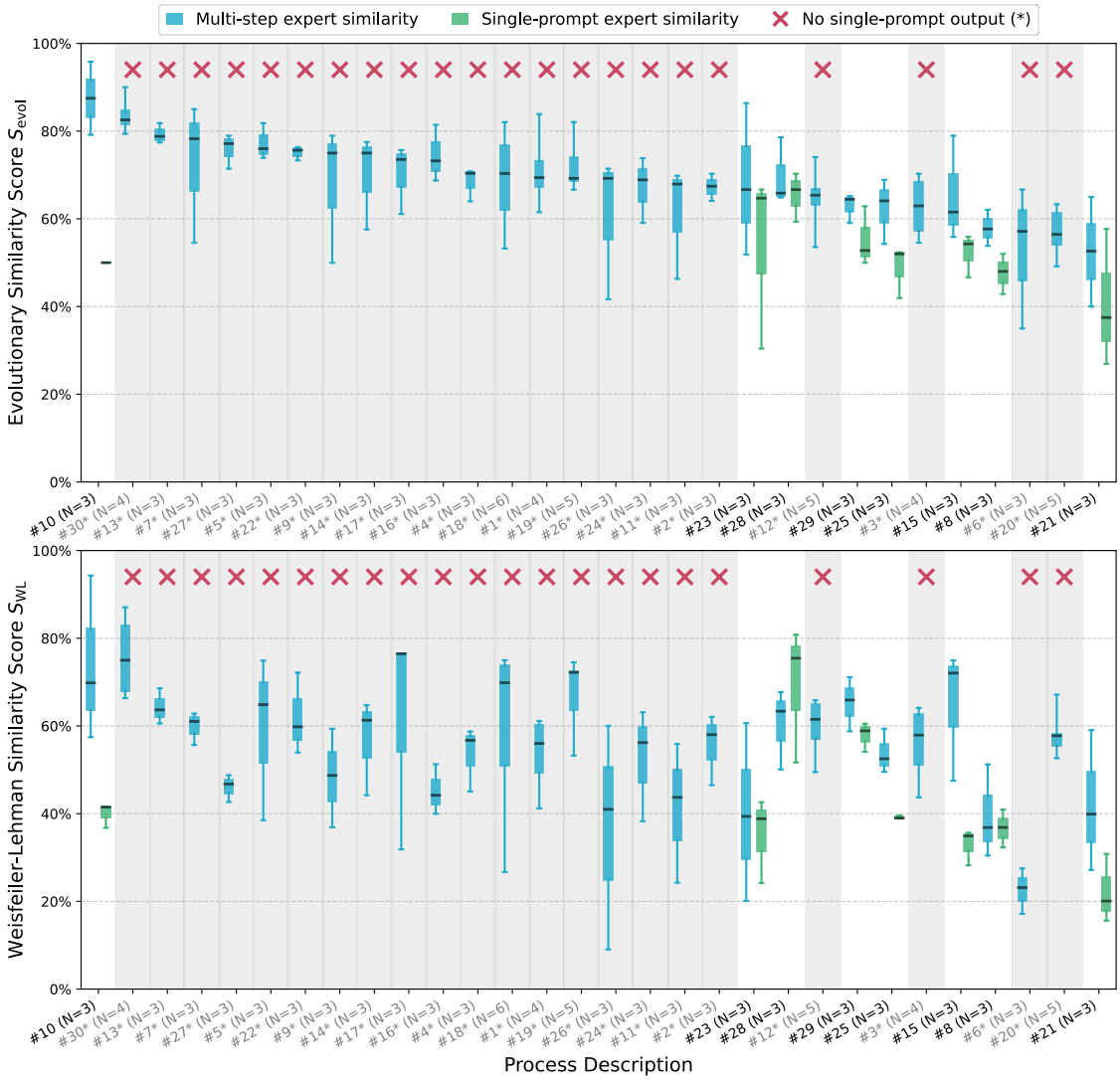


Figure 1: Similarity of LLM-generated flowsheets to the expert flowsheets using the text2flowsheet pipeline (blue) and using the single-prompt mode (green). If no green boxplot is shown, the single-prompt mode did not produce an adequate output for processing to a valid graph structure.

The shortcomings of the "single-prompt" setup motivate a central design decision of our multi-step pipeline: robust adherence to structured output by using smaller prompts with a limited scope each but high accuracy overall through their sequential execution. Furthermore, the multi-step pipeline can include steps outside the LLM prompt, such as rigorous thermodynamic computations, that aid the mechanistic grounding and interpretability of the flowsheet digitization. For these reasons, we would still recommend robust, structured, and grounded inference modes, even though LLMs advance rapidly and can perform increasingly complex tasks in a single prompt. Furthermore, from a user-experience perspective, the single- and multi-step modes appear very similar, as the "text2flowsheet" and "graph2simulation" tools can hide most of the multi-step complexity through their high degree of automation.

SI 6 Comparison between different LLMs

In the main manuscript, we used the cloud model GPT-5-mini whenever possible, so that users of the tool can easily obtain comparable results by plugging in their API key rather than setting up the local GPT-OSS model that requires substantial GPU resources. Figure 2 shows that the cloud and local models yield comparable results with a high degree of reliability. On average, the local model performs slightly better with 7.6 percentage points higher in the evolutionary metric, and 5.2 percentage points higher in the WL metric. The comparatively better performance should be seen in relation to ease of use, which for the default user is higher with cloud models. The examples detailed in the main manuscript, therefore, represent results in a quality that an average user can expect from deploying the data pipelines.

Figure 2: Comparison of LLM to expert similarities between the cloud model GPT-5-mini and the local model GPT-OSS.

We have further compared the performance of the much smaller, more consumer-grade local LLM Llama3.1 with 8 billion parameters to GPT-OSS. Figure 3 shows that the main failure mode is the lack of adherence to structured output (19 out of 30 cases), preventing further automated processing. However, when output is generated adequately, the digitized flowsheets are similar to the experts' interpretations, albeit less so than the flowsheets generated by the larger model.

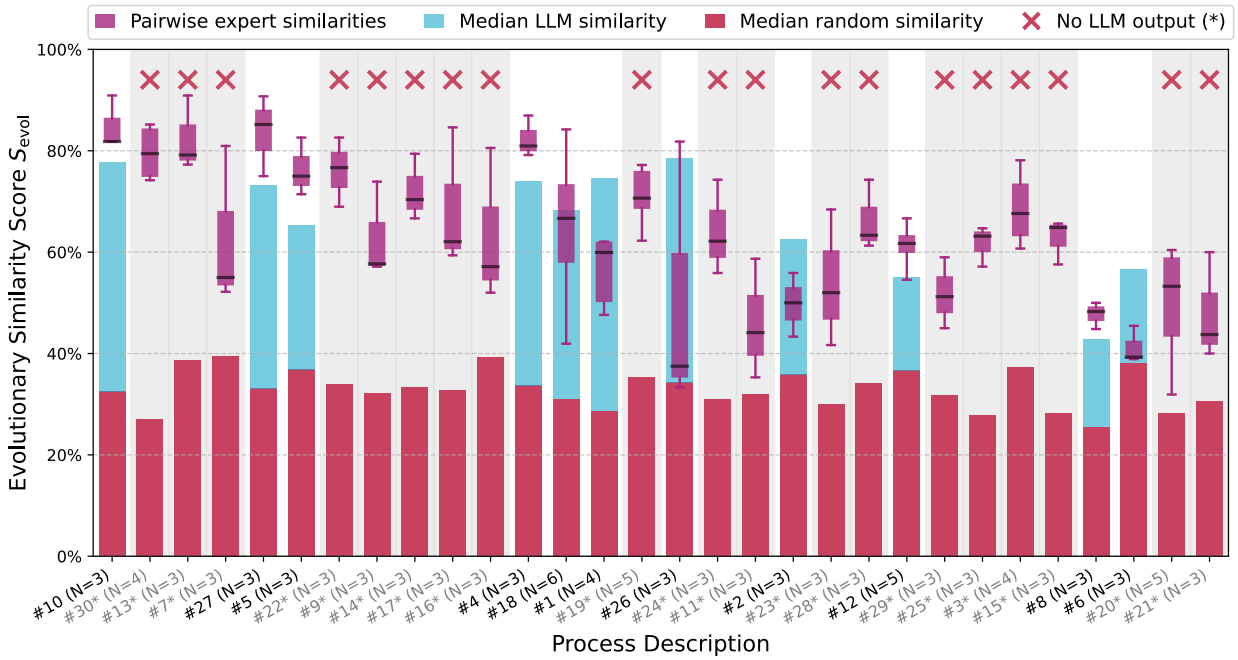


Figure 3: Performance of the text2flowsheet pipeline with respect to expert drawings using LLama3.1 8B. If no blue bar is shown, the pipeline did not produce an adequate output for processing to a valid graph structure.

SI 7 Permutation test on distinction between LLM and expert flowsheets

To test if the LLM-generated flowsheets are statistically indistinguishable from the expert-drawn flowsheets, we perform a permutation test for each process of the test set. Null hypothesis is $H_0 : T_{\text{LLM}} \sim T_{\text{reference}}$ where T_{LLM} is the mean similarity of the LLM graph to all expert graphs, and $T_{\text{reference}}$ is the same statistic computed for each expert graph to all other graphs. The alternative hypothesis is that the LLM graph is less similar compared to all expert graphs than when any expert graph is singled out and compared to all other graphs. Table SI 2 shows the p-values for each test set process under both similarity metrics discussed in manuscript Section 2.2.

Due to the low per-process sample size, we perform an aggregated meta-analysis over all 30 test processes using the Fisher combined p-value. For the evolutionary metric S_{evol}, the

combined p-value is 0.999718. For the kernel-based metric S_{WL}, the combined p-value is 0.999854. Overall, the null hypothesis cannot be rejected at any level, and there is no consistent trend of the LLM-generated graph being less similar than the expert graphs. Practically, the LLM-generated flowsheets are indistinguishable from the expert interpretations.

Table SI 2: p-values for the permutation test outlined above using both similarity metrics discussed in the main manuscript.

Process ID	$p(S_{\mathrm{evol}})$	$p(S_{\mathrm{WL}})$
#1	0.833	0.833
#2	0.800	0.600
#3	0.333	0.666
#4	0.400	0.400
#5	0.400	0.800
#6	1.000	1.000
#7	0.600	0.400
#8	1.000	0.800
#9	0.600	1.000
#10	0.600	0.600
#11	0.800	0.600
#12	0.857	0.571
#13	0.400	0.400
#14	0.400	0.400
#15	0.600	0.600
#16	0.333	0.333
#17	0.833	0.833
#18	0.555	0.666
#19	0.714	0.714
#20	0.857	0.857
#21	0.600	0.800
#22	0.600	0.400
#23	0.800	0.600
#24	0.666	1.000
#25	0.400	0.400
#26	0.600	1.000
#27	0.400	0.400
#28	0.555	0.400
#29	1.000	1.000
#30	0.833	0.833

SI 8 Full distribution of expert similarities

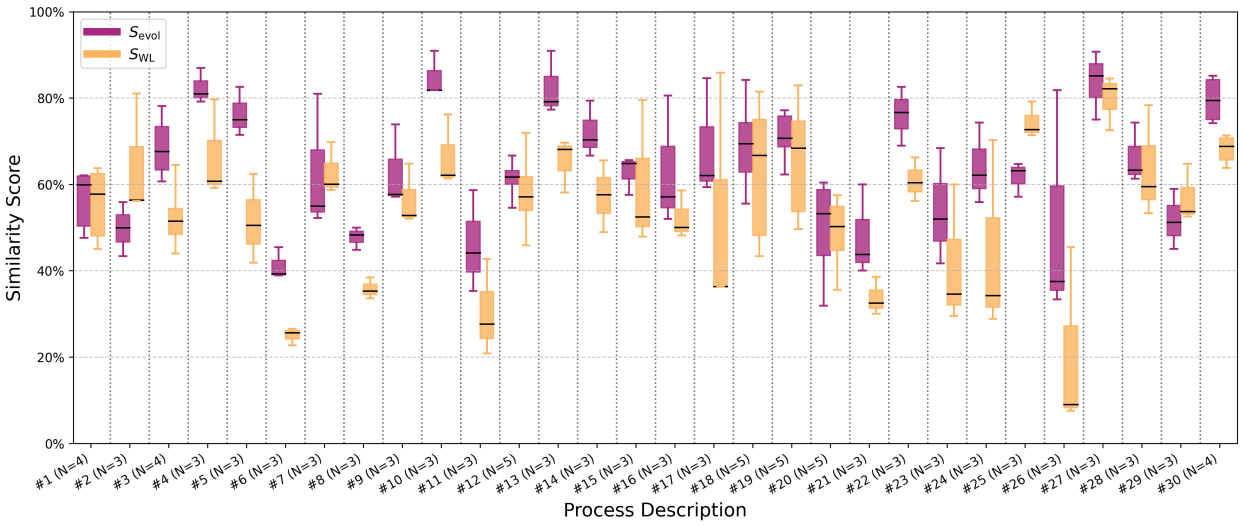


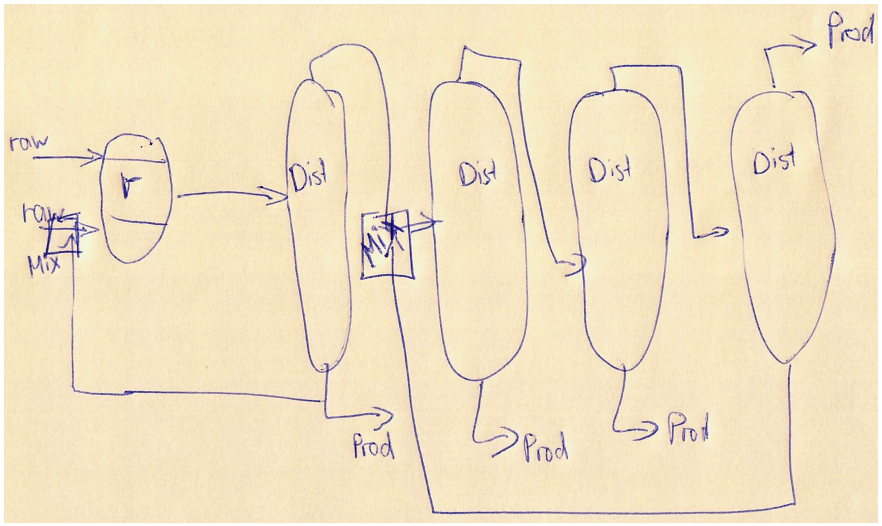
Figure 4: Distribution of pairwise expert similarities for all processes from the test set. N denotes the number of expert-drawn flowsheets for each test process.

SI 9 Expert drawing examples

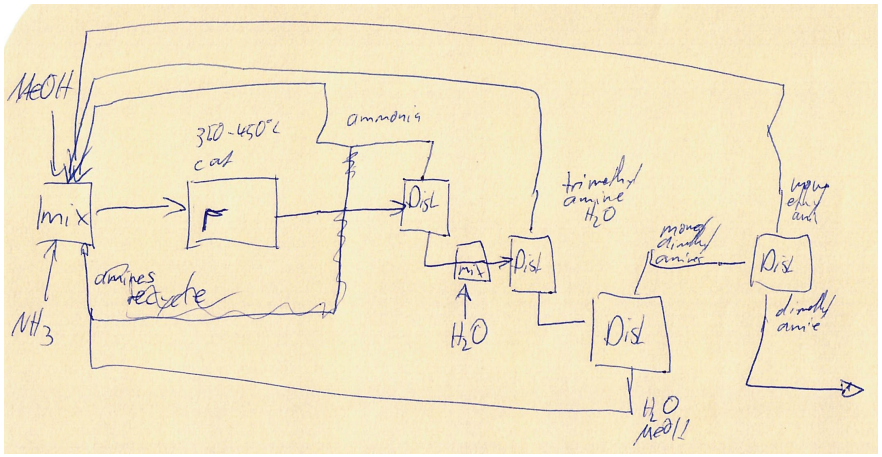
Examples of expert-drawn flowsheets from our collection campaign at the DECHEMA Annual Meeting of Process Engineering and Materials Technology 2025. These drawings were subsequently digitized by hand into the same graph format as the automatically digitized ones.

Process #5:

Expert 1



Expert 2



Expert 3

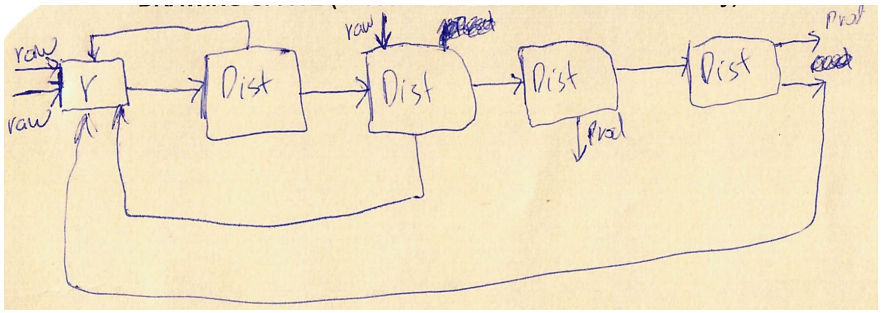
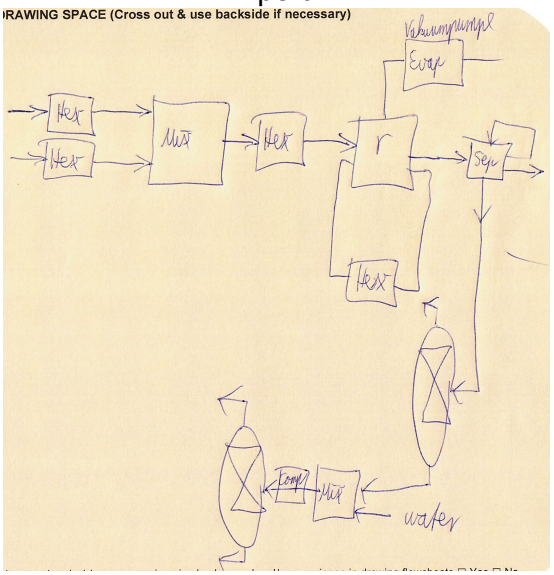


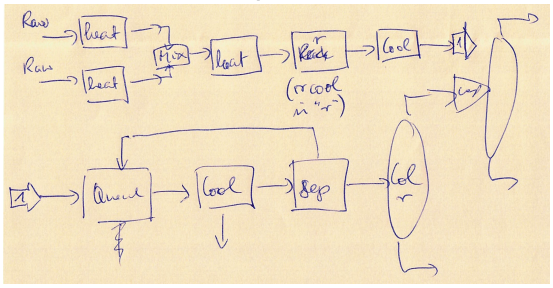
Figure 5: Expert-drawn flowsheet sketches for process #5 of the test set.

Process #18:

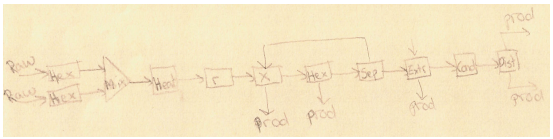
Expert 1



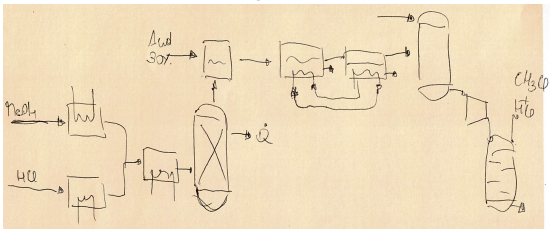
Expert 3



Expert 4



Expert 2



Expert 5

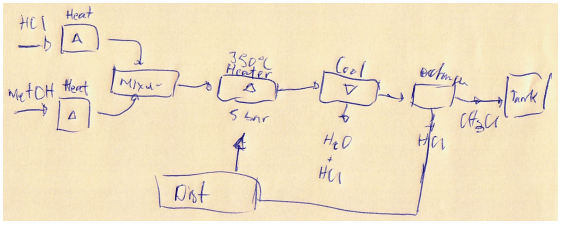


Figure 6: Expert-drawn flowsheet sketches for process #18 of the test set.

SI 10 Test set processes details

Table SI 3: Process text descriptions in test set. $|V|$ and $|E|$ denote the number of nodes (units, inlet and outlet streams) and edges (connecting streams) of the LLM-digitized flowsheet graph.

Process ID	Main product	Short description	Words in text	$\|V\|$ of graph	$\|E\|$ of graph	Source	LLM
#1	Acetaldehyde	VEBA-Chemie process for acetaldehyde from ethanol	123	17	15	Ullmann's[12]	gpt-5-mini
#2	Acetaldehyde	Wacker process for acetaldehyde from ethylene	197	23	23	Ullmann's	gpt-5-mini
#3	Formaldehyde	Formox process for formaldehyde from methanol	194	27	24	Ullmann's	gpt-5-mini
#4	Phosgene	Industrial phosgene process from carbon monoxide and chlorine	157	16	15	Ullmann's	gpt-5-mini
#5	Dimethylamine	Commercial process for dimethylamine from methanol and ammonia	176	19	20	Ullmann's	gpt-5-mini
#6	Propionaldehyde	Oxo synthesis process for propanal from ethene	194	19	20	Ullmann's	gpt-5-mini
#7	Aniline	Bayer process for aniline from nitrobenzene	188	22	22	Ullmann's	gpt-5-mini
#8	Aniline	Aniline production process from phenol	94	15	16	Ullmann's	gpt-5-mini
#9	Diisopropyl ether	Diisopropyl ether production from propylene	106	15	13	Ullmann's	gpt-5-mini
#10	Dimethyl malonate	Dimethyl malonate production from methyl chloroacetate	58	13	14	Ullmann's	gpt-5-mini
#11	Acetic Acid	BASF process for acetic acid from methanol	434	27	34	Ullmann's	gpt-5-mini
#12	2-Nitrotoluene	Nitrotoluene production from toluene	211	19	17	Ullmann's	gpt-5-mini
#13	Catechol	Rhône-Poulenc process for catechol from phenol	171	18	18	Ullmann's	gpt-5-mini
#14	Formaldehyde	Partial oxidation process for formaldehyde from methanol	482	28	28	Ullmann's	gpt-5-mini
#15	Nitrobenzene	Continuous nitrobenzene process from benzene	196	21	24	Ullmann's	gpt-5-mini
#16	1,4-Butanediol	Industrial production of 1,4-butanediol from 2-butyne-1,4-diol	136	22	16	Ullmann's	gpt-5-mini
#17	Monoethylene Glycol	Monoethylene glycol production process from ethylene oxide	210	19	17	Ullmann's	gpt-5-mini
#18	Methyl Chloride	Methyl chloride production from methanol	373	27	24	Ullmann's	gpt-5-mini
#19	Dichloromethane	Hoechst process for dichloromethane from methanol	275	23	22	Ullmann's	gpt-5-mini
#20	HCFC-141B	HCFC-141B production process from methyl chloroform	247	33	37	IHS PEP[13]	gpt-oss
#21	Butadiene	Butadiene production from n-butane	191	21	21	IHS PEP	gpt-oss
#22	Butadiene	Butadiene production from butenes	125	19	15	IHS PEP	gpt-oss
#23	Hexene-1	Alphahexol process for 1-hexene from ethylene	157	11	12	IHS PEP	gpt-oss
#24	Hydrogen cyanide	Andrussow process for hydrogen cyanide from methane	180	17	18	IHS PEP	gpt-oss
#25	Maleic anhydride	Maleic anhydride production from n-butane	172	16	17	IHS PEP	gpt-oss
#26	Methyl isobutyl carbinol	Methyl isobutyl carbinol production from methyl isobutyl ketone	60	8	5	IHS PEP	gpt-oss
#27	Phenol	Phenol production from benzene	163	18	20	IHS PEP	gpt-oss
#28	Acetic acid	Low pressure acetic acid production from methanol	249	21	21	IHS PEP	gpt-oss
#29	Ethylene	Ethylene production from ethene	166	22	20	IHS PEP	gpt-oss
#30	Acrylic acid	Nippon Shokubai process for acrylic acid from propylene	158	13	14	IHS PEP	gpt-oss

SI 11 Objective functions for black-box unit optimization

Note that the optimizer can only adjust specific model parameters $\mathbf{p}$, e.g., the reflux ratio or reactor temperature, and cannot downgrade the model itself. This model simplification is kept outside the optimization loop to avoid that the optimizer exploits simplifications to artificially reduce the energy demand.

HK = heavy key component

LK = light key component

P = product component

E = extracted component

R = raffinate component

flash

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{LK in liquid}}}{\dot{m}_{\text{HK in liquid}}} + \frac{\dot{m}_{\text{HK in vapor}}}{\dot{m}_{\text{LK in vapor}}} & \{\text{ensuring separation split efficacy}\} \\ |\dot{Q}| & \{\text{preventing excessive energy demand}\} \end{pmatrix} \tag{3}$$

Decanter

$$\min\ f(\mathbf{p}) = \frac{\dot{m}_{\text{LK in heavy phase}}}{\dot{m}_{\text{HK in heavy phase}}} + \frac{\dot{m}_{\text{HK in light phase}}}{\dot{m}_{\text{LK in light phase}}} \tag{4}$$

RGibbs & RStoic

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} -\frac{\dot{m}_{\text{P}}}{\dot{m}_{\text{Reactor outlet}}} \\ |\dot{Q}| \end{pmatrix} \tag{5}$$

Radfrac (distillation) & DSTWU

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{LK in bottom}}}{\dot{m}_{\text{HK in bottom}}} + \frac{\dot{m}_{\text{HK in top}}}{\dot{m}_{\text{LK in top}}} \\ -\dot{Q}_{\text{Cond}} + \dot{Q}_{\text{Reb}} \end{pmatrix} \tag{6}$$

Condenser

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{vapor}}}{\dot{m}_{\text{liquid}}} \\ |\dot{Q}| \end{pmatrix} \tag{7}$$

Evaporator

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{liquid}}}{\dot{m}_{\text{vapor}}} \\ |\dot{Q}| \end{pmatrix} \tag{8}$$

Extr

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{E in raffinate stream}}}{\dot{m}_{\text{R in raffinate stream}}} + \frac{\dot{m}_{\text{R in extract stream}}}{\dot{m}_{\text{E in extract stream}}} \\ \dot{m}_{\text{solvent}} \end{pmatrix} \tag{9}$$

Radfrac (absorption)

$$\min\ \mathbf{f}(\mathbf{p}) = \begin{pmatrix} \frac{\dot{m}_{\text{E in vapor}}}{\dot{m}_{\text{R in vapor}}} + \frac{\dot{m}_{\text{R in liquid}}}{\dot{m}_{\text{E in liquid}}} \\ \dot{m}_{\text{solvent}} \end{pmatrix} \tag{10}$$

SI 12 Example of multi-object black-box optimization

The optimization of the "ACOLUMN" in example 3 of the main manuscript is performed with respect to two objective functions: the separation split objective, i.e., minimizing the fraction of heavy key in the top phase plus light key in the bottom phase, and the column's aggregated energy demand. Figure 7 shows the obtained Pareto curve. As the final configuration, a point is chosen that minimizes the aggregated energy demand within a small tolerance region around the point with the best separation effectiveness. We have seen that this heuristic tends to yield more reasonable configurations that achieve the desired separation while avoiding excessive energy consumption. This heuristic, however, is intended only as a starting point and could be refined through expert-driven, process-specific analysis and optimization.

SI 13 Process descriptions of examples 1, 2, and 4

SI 13.1 Example 1

"One of the oldest production methods is that of Hoechst, a recycle chlorination which was introduced as early as 1923 and which, apart from modifications reflecting state-of-the-art technology, continues essentially unchanged, retaining its original importance. The gas which is circulated consists of a mixture of methane and monochloromethane. To this is added fresh methane and, as appropriate, monochloromethane obtained from methanol hy-

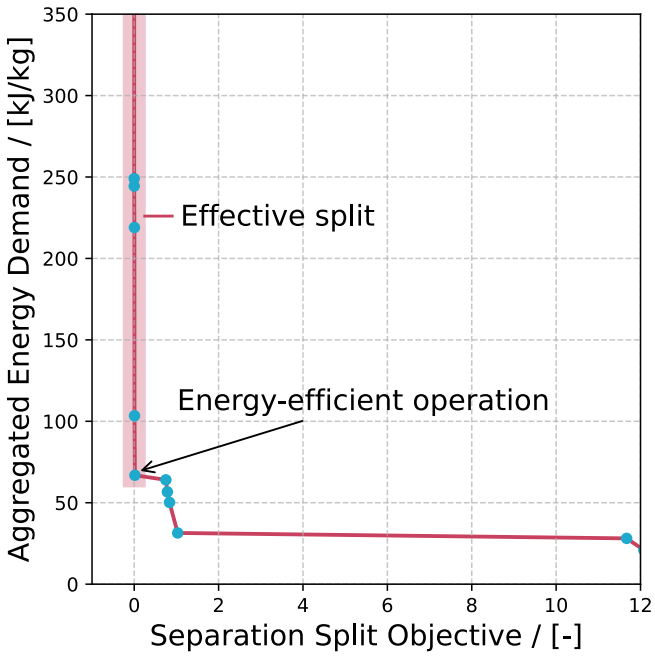


Figure 7: Pareto curve of separation split objective and aggregated energy demand (heating plus cooling) for the ammonia separation column ("ACOLUMN") of the aniline from phenol process. The energy demand is normalized with respect to the incoming mass flow.

drochlorination. Chlorine is then introduced and the mixture is passed into the reactor. The latter is a loop reactor coated with nickel or highalloy steel in which internal gas circulation is constantly maintained by means of a coaxial inlet tube and a valve system. The reaction is conducted adiabatically, the necessary temperature of 350 – 450 °C being achieved and maintained by proper choice of the chlorine to starting material (CH4 + CH3Cl) ratio and/or by prewarming the mixture. The fully reacted gas mixture is cooled in a heat exchanger and passed through an absorber cascade in which dilute hydrochloric acid and water wash out the resulting hydrogen chloride in the form of 31 % hydrochloric acid. The last traces of acid and chlorine are removed by washing with sodium hydroxide, after which the gases are compressed, dried, and cooled and the reaction products largely condensed. Any uncondensed gas — methane and to some extent monochloromethane — is returned to the reactor. The liquified condensate is separated by distillation under pressure into its pure components, monochloromethane, dichloromethane, trichloromethane (the latter two being

the principal products), and small amounts of tetrachloromethane. The product composition is approximately 70 wt % dichloromethane, is approximately 27 wt % trichloromethane, and 3 wt % tetrachloromethane." Rossberg *et al.* (2011)[14]

SI 13.2 Example 2

"*Bayer* operates conventional fixed-bed reactors using a palladium catalyst on a alumina support, modified in its activity by the addition of vanadium and lead and claims the adiabatic hydrogenation of nitrobenzene over a fixed-bed catalyst of 1.5 to 4 wt % palladium on coke with 0.1 to 2 wt % lead as a modifier to reduce aromatic ring hydrogenation. At a pressure of 100 – 700 kPa a mixture of vaporized nitrobenzene and hydrogen in a molar ratio of 1 : 120 to 1 : 200 is fed to the adiabatic reactor with an inlet temperature of 250 – 350 °C. The height of the catalysts' bed in the reactor is 0.1 to 1.0 m. The reaction products leave the reactor without cooling at a maximum temperature of 460 °C. After leaving the reactor, the heat of reaction is used for the production of high pressure steam. A production unit can be built of several serial and/or parallel adiabatic reactors. After cooling down to 140 to 180 °C, the outlet of the final reactor is fed to a separation unit, where after further cooling crude aniline, crude wastewater, and the recycled hydrogen are separated under pressure. The crude aniline is purified by distillation." Kahl *et al.* (2011)[15]

SI 13.3 Example 4

"Carbon monoxide, methanol (containing up to 60% dimethyl ether), catalyst recycle, catalyst makeup, and methyl iodide recycle (from the wash column) are sent to the high-pressure reactor (stainless steel lined with Hastelloy). Part of the relatively low heat of reaction is used to preheat the feed and the rest is ultimately dissipated through the reaction vent. The reaction product is cooled and sent to the high-pressure separator. The off-gas goes to the wash column and the liquid is expanded to a pressure of 0.5–1.0 MPa in the intermediate-pressure separator. The gas released is also sent to the wash column; the liquid from the

intermediate-pressure separator is sent to the expansion chamber. The gas from the chamber goes to the scrubber. The gas from the scrubber and the wash column is discarded as off-gas. Both scrubber and wash column use the methanol feed to recover methyl iodide and other iodine-containing volatile compounds; this methanolic solution is returned to the reactor. The off-gas composition in vol% is 65–75 CO, 15–20 CO2, 3–5 CH4, and the balance CH3OH. The raw acid from the expansion chamber contains 45 wt% acetic acid, 35 wt% water, and 20 wt% esters, mainly methyl acetate. The acid is purified in five distillation towers. The first column degasses the crude product; the off-gas is sent to the scrubber column. The catalyst is then separated as a concentrated acetic acid solution by stripping the volatile components in the catalyst separation column. The acid is then dried by azeotropic distillation in the drying column. The overhead of the drying column contains acetic and formic acids, water, and byproducts that form an azeotrope with water. This overhead is a two-phase system that is separated in the chamber. Part of the organic phase, composed mainly of esters, is returned to column, where it functions as an azeotroping agent. The remainder of the organic phase is sent to the auxiliary column where heavy ends are separated at the bottom of the column, and light esters from the overhead are recycled to the reactor. The aqueous phase and the catalyst solution are returned to the reactor. The base of the drying column is sent to a finishing column, in which pure acetic acid is taken overhead. The bottom stream of the finishing tower is sent to the residue column. The overhead of this residue column is sent back to the dehydration column. The bottom of the residue column contains about 50 wt% propionic acid, which can be recovered." Le Berre *et al.* (2014)[16]

SI 14 LLM prompts for "text2flowsheet"

The LLM prompts themselves have been partially formulated and restructured using a large language model.

SI 14.1 Topology construction

Unit operation extraction:

```
class UnitExtraction(dspy.Signature):
    """
    You are a highly accurate extraction engine for chemical process modeling.

    Your task is to identify all unit operations from a given process description.

    Your job is to:

    1. Carefully read the process description.
    2. Identify every instance where a unit operation is either explicitly mentioned (e.g., "a mixer", "the distillation column") or implied based on function (e.g., "two streams are combined", which implies a Mixer).
    3. Give every unit an individual name related to their role in the process.
    4. Specify the unit with a natural-language description paraphrased from the process text. **Do not mention any chemical in this description that is not clearly and explicitly mentioned in the process description for that unit or stream.**

    5. Assign the unit operation a type based on the following list. Use only the abbreviations.
    6. Determine if the unit is involved in the operative procedures of the process to produce and refine the target chemical(s), or solely in the maintenance of the reactor or catalyst that happens outside the regular production sequence.

    Return a list of `UnitOperation` objects, one for each extracted unit.
```

```
18
19    Only use the following list of allowed unit operation types and
      abbreviations:
20
21    Absorption | Abs
22    Centrifuge | Centr
23    Compressor | Comp
24    Condenser | Cond
25    Cooler | Cool
26    Cyclone | Cycl
27    Distillation (incl. reboiler and condenser) | Dist
28    Drying | Dry
29    Evaporator | Evap
30    Expander | Expand
31    Extraction | Extr
32    Extractive Distillation | exDist
33    Flash | flash
34    Gas filtration | Gfil
35    Heater | Heat
36    Heat exchanger | Hex
37    Hydrocyclone | Hcycl
38    Liquid filtration | Lfil
39    Mixing | Mix
40    Pump | pp
41    Reactor | r
42    Rectification | Rect
43    Scrubbing | Scrub
44    Separation (no further sub-specification) | Sep
45    Settling | Set
46    Splitting | Splt
47    Stripping | Strip
48    Storage | Tank
49    Unknown unit | X
```

```
50
51      **Be thorough and exhaustive.** Every step or stage in the process
        should be treated as a unit operation, even if the description is vague
        or lacks detail.
52      Regarding the unit type: When uncertain, make the best possible
        inference based on context and your knowledge of process engineering
        and use the types 'X' (Unknown unit) and 'Sep' (unspecified separation)
        only when absolutely necessary.
53
54      Extract condensers only when they are separate from a distillation, i.
        e., they condense a stream that is not coming out of a distillation
        column. Otherwise, only extract the distillation unit.
55
56      Only add splitters for recycle streams if one is **explicitly**
        described in the process description.
57
58      Return the results as a list of unit operation objects with the
        following fields:
59      - name: a short name of the unit
60      - type: one of the abbreviations listed above
61      - description: a concise explanation of the units purpose and
        features
62      - function: the main function that the unit is involved in: either "
        production", "maintenance", or "unclear"
63      """
64      process_description: str = dspy.InputField()
65      unit_operations: List[UnitOperation] = dspy.OutputField(desc="List of
        UnitOperation objects of the specified type found in the process.")
```

Raw material stream extraction:

```
1   class RawExtraction(dspy.Signature):
2       """
3       You are a highly accurate natural language processing engine
```

```
    specialized in the analysis of chemical process descriptions.
    Your task is to extract and describe **all raw materials** used in the
    input process description.

    **Be thorough and exhaustive.** Identify every material that enters
    the process from **outside its described boundaries**, whether it's a
    reagent, solvent, feedstock, catalyst, gas, or other input. **Do not**
    include intermediates formed within the process or materials recycled
    internally.

    When uncertain, make the best possible inference based on context, but
    only include streams **explicitly described** in the process
    description. Do not include utility streams for heating and cooling.

    For each identified raw material:
    - Provide the common or chemical name of the material.
    - Write a brief description of the materials role or function in
    the process (e.g., primary reactant, solvent, inert gas, etc.).
    """
    process_description: str = dspy.InputField()
    raw_materials: List[RawMaterial] = dspy.OutputField(
        desc="List of objects describing the raw materials of the process.
    "
    )
```

Product stream extraction:

```
class ProductExtraction(dspy.Signature):
    """
    You are a highly accurate natural language processing engine
    specialized in analyzing chemical process descriptions.

    Your task is to determine the final outputs of the process and how
    they are handled.
```

```
6
7    Perform this in two steps:
8
9    **Step 1: Identify and describe all final materials at the end of the
     process.**
10   These may include:
11   - Intended products (main products, co-products)
12   - Unintended byproducts
13   - Leftover or unconverted reactants
14   - Waste streams or purge gases
15
16   Be comprehensive. If the process ends with a stream that contains
     multiple components not separated in the flowsheet, treat the mixture
     as a single product unless there's clear indication of separation.
17
18   **Step 2: For each identified material, determine if it leaves the
     process or is recycled.**
19   - Mark `leaves_process=True` if the material exits the process
     boundary (as a product, waste, purge, etc.).
20   - Mark `leaves_process=False` if it is returned to an earlier unit or
     reused internally (i.e., part of a recycle loop or closed system).
21
22   **Do not** include intermediates or circulating internal streams
     unless they are explicitly described as leaving the process.
23
24   **Do not** invent products. Only extract materials that are explicitly
     or implicitly described in the process description.
25
26   Return a list of `Product` objects, each with:
27   - `name`: The chemical or common name of the material.
28   - `description`: A short explanation of its role or context in the
     process.
29   - `leaves_process`: Whether it exits the process or is recycled
```

```
    internally.
30      """
31      process_description: str = dspy.InputField()
32      products: List[Product] = dspy.OutputField(
33          desc="List of objects describing the final outputs of the process
    and whether they leave or are recycled."
34      )
```

Determine subsequent unit at open stream in graph:

```
1  class NextUnitQuery(dspy.Signature):
2      """
3      You are a chemical process flowsheet expert analyzing the structure of
       the flowsheet underlying the given process description.
4
5      Given the current unit operation and a specific outgoing stream label,
       determine the identifier (unit_id) of the next downstream unit
    operation stream connected to this stream or connect it to a final
    product stream.
6      Always choose the immediately following process operation, i.e., when
    two reactants are first mixed and then reacted, the following unit
    operation would be a mixer, or when a stream is first heated / cooled /
       pumped / compressed and then further processed, the following unit
    would be a heater / cooler / pump / compressor.
7
8      The outgoing stream label gives you a hint about the connection we
    want to establish with respect to the current unit operation. "main"
    means, that the current unit operation has only one outlet, so we are
    simply looking for the next unit operating that receives this outgoing
    stream. "top" / "bottom" indicates that we are looking at the top /
    bottom stream of the current unit (a distillation column), which gives
    you a hint which following unit we want to extract from the process
    description. Similarly, "vapor" / "liquid" indicate the two different
    outgoing streams of a flash, "light" / "heavy" of a settler etc.
```

```
9     All units except final product outlet streams have such labels,
      indicating that a connection must exist. Only final product outlet
      streams have no further connections.
10    In the case of two outgoing stream labels, make an educated guess
      which label corresponds to which stream from the text. If one of the
      stream labels is already connected to a unit operation, infer from that
       connection which other stream from the text could be represented by
      the to-be-determined label.
11    Be particularly aware of units that result in the final product and
      make sure that one of its outgoing connections leads to a "Prod" node
      of that final product.
12
13    The current unit can also be a raw material (indicated by the label "
      raw"). This signifies an incoming stream into the process boundaries
      and we are looking for its first processing step.
14
15    If the current unit is a tank or storage unit (also indicated by the
      label "store"), the following unit will be a product stream (type == "
      Prod"). This might seem unconventional, but in this case it signifies
      that in principle, product can be taken out of this storage and thus
      leaving the overall process boundary.
16    Product streams may also follow other unit operations, e.g., usually
      after separations or other purification steps.
17
18    When the description mentions that a stream is "returned", "recycled",
       or something similar, the following unit is situated earlier in the
      process flow, close to the current unit. Be thorough when examining
      possible following units for recycled streams and choose one that is
      situated BEFORE the current unit and not one in a parallel branch of
      the flowsheet. Do not be confused by similar sounding unit names or
      descriptions!
19
20    You have access to the entire process description and the full list of
```

```
    all units that was extracted from this description and labeled before.
21
22      Included in the list of all units are also the final product streams
    leaving the process, which can be selected as following unit operations
     as well.
23
24      **Some unit operations may be explicitly excluded from selection in a
    given query. These are listed in the `excluded_units` field. You must
    not select any unit_id from this list as the next downstream unit. When
     in doubt, rather return None instead of a unit from the list of
    excluded units.**
25
26      Adhere to the given process description under any circumstance, even
    if common engineering knowledge suggests otherwise.
27
28      When uncertain, make the best possible inference based on context.
    When no direct following unit is apparent, first reexamine the units
    with the type == "Prod", as the stream from the current unit may leave
    the system as a product, by-product, or waste stream.
29      Always try to find a following unit operation, but if the outgoing
    stream is leaving the scope of the process description, return None.
30
31      Respond with the next unit's identifier (e.g., "U_3"), a single string
    .
32      """
33      process_description: str = dspy.InputField()
34      unit: UnitWithConnections = dspy.InputField()
35      output_label: str = dspy.InputField()
36      all_units: List[UnitWithConnections] = dspy.InputField()
37      excluded_units: List[UnitWithConnections] = dspy.InputField()
38
39      next_unit_id: Union[None,constr(pattern=r"^U_\d+\$")] = dspy.
    OutputField(
```

```
        desc="The unit_id of the next downstream unit."
    )
```

Integrate leftover unit in graph:

```
class DetermineUpstreamUnit(dspy.Signature):
    """
    You are a chemical process flowsheet expert.

    Your primary goal is to determine where the candidate unit most
    logically fits into the existing process sequence.

    You are given:
    - A description of the overall process.
    - The candidate unit that has not yet been placed.
    - A list of units that are already connected in the process, along
    with their known connections.

    Your task is to identify the upstream unit that the candidate most
    likely follows. Focus on logical placement within the process narrative
     and functional flow. Stick to the process description at all times.

    Use all available information - including the candidate units name,
     type, and description, as well as clues from the process description
        to infer its likely upstream context.

    Make your best educated guess, and only return None if no logical
    upstream unit based on the process description can be identified.

    Return:
    - The `unit_id` of the upstream unit that should precede the candidate
    , if a confident placement can be made.
    - Return `None` if no suitable upstream unit can be identified.
    """
```

```
    process_description: str = dspy.InputField()
    candidate_unit: UnitWithConnections = dspy.InputField()
    connected_units: List[UnitWithConnections] = dspy.InputField()

    upstream_unit_id: Union[None, constr(pattern=r"^U_\d+$")] = dspy.
    OutputField(
        desc="The unit_id of the upstream unit that the candidate unit
    should follow."
    )
```

Integrate leftover product stream in graph:

```
class DetermineFinalProductUpstream(dspy.Signature):
    """
    You are a chemical process flowsheet expert.

    Your task is to determine which unit in the process produces a
    specific product, byproduct, or waste stream.

    You are given:
    - A natural language description of the overall process.
    - The candidate unit representing the product, byproduct, or waste
    stream.
    - A list of connected units in the process and their current outgoing
    connections.

    Focus your analysis on identifying the upstream unit that generates
    the candidate unit stream. Specifically:

    1. Look for units with **undefined (‘None‘) outgoing connections**, as
    these may indicate yet undetermined streams that need a destination.
    2. Consider context from the process description that indicates the
    stream is leaving the process, such as mentions of collection, disposal
    , or transfer.
```

```
    3. Do **not** connect the candidate unit mid-process or into a recycle loop.

    Make the best educated guess based on the process description.

    Return:
    - The `unit_id` of the upstream unit that produces this stream.
    """
    process_description: str = dspy.InputField()
    candidate_unit: UnitWithConnections = dspy.InputField()
    connected_units: List[UnitWithConnections] = dspy.InputField()

    upstream_unit_id: Union[constr(pattern=r"^U_\d+$")] = dspy.OutputField(
        desc="The unit_id of the upstream unit that produces the finalized product stream to be handled by the candidate unit."
    )
```

Helper prompt to determine possible upstream connections of leftover units:

```
def make_determine_connection_labels_signature(upstream_unit, candidate_unit):
    upstream_keys = list(upstream_unit.following_units.keys())
    candidate_keys = list(candidate_unit.following_units.keys())

    # Safeguard: if either set is empty, provide a dummy value to avoid Literal[()] error
    upstream_literals = tuple(upstream_keys) if upstream_keys else ("_no_valid_labels_",)
    candidate_literals = tuple(candidate_keys) if candidate_keys else ("_no_valid_labels_",)

    class DetermineConnectionLabels(dspy.Signature):
        """
```

```
    You are a chemical process flowsheet expert.

    Your task is to help integrate a new (candidate) unit into an existing chemical process flowsheet by determining how it fits both logically and structurally.

    You are given:
    - A process description.
    - A candidate unit that is not yet connected.
    - An upstream unit that the candidate has been determined to follow.

    Your objective is to determine how this new unit should be connected in the process stream:

    1. Based on the process description and unit metadata (type, name, description), select the output label from the upstream unit through which it should connect to the candidate unit. This label must be one of the keys in `upstream_unit.following_units`. If the candidate unit has the type "Prod", always choose a label that is unconnected in the upstream unit (i.e., label: None), or return None if the upstream unit has no unassigned connections. If the candidate unit has any other type and the label assignment is unclear, also preferably choose the unconnected label.

    2. If the upstream unit was already connected to a downstream unit through that label, select the output label on the candidate unit (from its `following_units`) that should now be used to continue the stream to that downstream unit.

    In doing so, you are refining and maintaining the correct process flow, ensuring that stream logic is preserved while inserting the new unit.
```

```
27
28          Return `None` for `handoff_label` if no downstream unit is displaced.
29          """
30          process_description: str = dspy.InputField()
31          upstream_unit: UnitWithConnections = dspy.InputField()
32          candidate_unit: UnitWithConnections = dspy.InputField()
33
34          connection_label: Union[None, Literal[upstream_literals]] = dspy.OutputField(
35              desc="The output label on the upstream unit used to connect to the candidate unit. Must be in `upstream_unit.following_units`."
36          )
37          handoff_label: Union[None, Literal[candidate_literals]] = dspy.OutputField(
38              desc="The output label on the candidate unit to forward the stream to a downstream unit (if one was displaced). Must be in `candidate_unit.following_units`."
39          )
40
41      return DetermineConnectionLabels
```

Characterize added outlet stream:

```
1 class InferMissingOutputStream(dspy.Signature):
2     """
3     You are a highly accurate reasoning engine for reconstructing missing output streams in chemical process descriptions.
4
5     Your task is to infer the identity and purpose of a **missing output stream** from a given unit operation. This occurs when a unit has a defined output label but no downstream connection, indicating an unhandled flow leaving the system.
6
```

```
    You are given:
    - The full process description.
    - A specific unit (‘unit‘) that has a missing outflow.
    - The name of the output label from which the stream leaves (e.g., "bottom", "light", "raffinate", "filtrate").

    Your job is to:
    1. Analyze the process context and the role of the unit to determine what stream is expected to exit through this label.
    2. Infer the most likely **purpose** and **contents** of the stream (e.g., product, waste, purge, vent, bleed, recycle).
    3. Generate:
        - A **short, meaningful name** for a new stream unit that will receive this output (this name will become a new ‘Prod‘ unit).
        - A **brief description** (1–2 sentences) explaining what the stream is, where it comes from, and why it is present in the process.

    Guidelines:
    - Base your inference on the process description, stream label, and type/purpose of the unit.
    - Do not leave any fields blank. Even in ambiguous cases, provide your best justified guess based on chemical process engineering logic.

    Return:
    - ‘name‘: A concise identifier for the new output stream.
    - ‘description‘: A short explanation of what the stream represents and its role in the process.

    This task is required for every unconnected output. Do not return None.
    """
    process_description: str = dspy.InputField()
    unit: UnitWithConnections = dspy.InputField()
```

```
    connection_label: str = dspy.InputField()

    name: str = dspy.OutputField(desc="Specific name for the new output stream unit")
    description: str = dspy.OutputField(desc="Concise but detailed explanation of what this stream is and why it exists.")
```

Characterize added inlet stream:

```
class ExtractionMediumInference(dspy.Signature):
    """
    You are a chemical process analyst specializing in separation and extraction steps.

    Your task is to decide whether a given unit operation requires the presence of an extraction medium or solvent,
    and if so, to identify the most likely chemical that serves this purpose.

    You are given:
    - The complete text of the process description.
    - The specific unit operation under consideration (usually of type Extr, Scrub, Wash, Leach, Abs, Stripping, etc.).

    Guidelines:
    - First decide if an extraction medium or solvent is required for this unit.
    - If required, propose the single most likely solvent or medium by its **explicit name**.
    - Prefer precise chemical names or standard trade names over vague descriptors.
    - Do not invent exotic solvents unless strongly implied; prefer common industrial solvents (e.g., water, toluene, hexane) when supported by the context.
```

```
    - If no solvent is needed, set `needs_medium=False` and leave the `medium_name` field as None.

    Return:
    - `needs_medium`: True if the unit requires a solvent/extraction medium, otherwise False.
    - `medium_name`: The name of the solvent (string) if one is needed, otherwise None.
    """

    process_description: str = dspy.InputField()
    extraction_unit: UnitWithConnections = dspy.InputField()

    needs_medium: bool = dspy.OutputField(
        desc="True if the unit requires an extraction medium/solvent, otherwise False."
    )
    medium_name: Optional[str] = dspy.OutputField(
        desc="The most likely solvent/extraction medium required by this unit (string), or None if not needed."
    )
```

SI 14.2 Data augmentation

Extract chemical compounds in inlet or outlet stream:

```
class StreamCompoundExtraction(dspy.Signature):
    """
    You are a highly accurate chemical name extractor for process stream analysis.

    Your task is to extract the **chemical compound(s)** present in a given stream from a process flowsheet.

```

```
    The stream is represented by a unit operation of type "Prod" (product leaving the system) or "Raw" (material entering the system). These streams typically contain **one** main chemical component, but in some cases, they might contain more.

    You are given:
    - The full process description.
    - The specific stream unit (of type "Prod" or "Raw") to analyze.

    **Your job is to identify the chemical compound(s) that are clearly present in this stream.**

    Be highly specific and concise:
    - Only provide one name per chemcial! Choose the primary name given for any component. Prefer explicit trade names over chemical names.
    - **Avoid vague names** (e.g., "residue", "product stream") or generic descriptors.
    - Do **not** include process roles or descriptions     just return the clean compound names.
    - Be exhaustive but precise: only list compounds that are **explicitly or clearly implied** to be present in the stream.

    When uncertain, make the best possible inference from the surrounding context in the process description.

    Return a list of StreamCompound objects.
    """
    process_description: str = dspy.InputField()
    stream_unit: UnitWithConnections = dspy.InputField()

    compounds: List[StreamCompound] = dspy.OutputField(
        desc="A list of clearly named chemical compounds found in this stream, suitable for conversion to SMILES."
```

```
    )
```

Interpret ambiguous compound name:

```
class InterpretAmbiguousCompound(dspy.Signature):
    """
    You are a chemical informatics expert with deep domain knowledge of industrial and organic chemistry.

    Your task is to interpret a single unresolved or ambiguous compound name that could not be resolved by PubChem or other canonical sources.

    These names often include:
    - Trade names, process-specific terms (e.g., "light ends", "reaction residue", "sour water")
    - Chemical classes or shorthand (e.g., "sodium salts", "MEG", "glycols")
    - Mixtures, side streams, or byproducts whose composition depends on the reaction context

    You are provided with the full chemical process description from which this name was extracted.

    Your goals are to:
    1. **Infer what the name likely refers to chemically**, using your background and the process context (e.g., type of reaction, reagents, known co-products).
    2. **Propose one or more plausible chemical components** that best match the name. Use IUPAC or widely accepted names.
    3. **Break down mixtures when appropriate**, identifying major components.
    4. Be **chemically conservative**  only unify ambiguous terms if the inference is strongly supported by the context.
    5. Preserve **case distinctions and abbreviations** if they are
```

```
    chemically meaningful (e.g., "MEA"     "Mea").
    6. Return a concise explanation of how you interpreted the name.

    Input:
    - A single `original_name` that failed resolution
    - The full `process_description` for contextual reasoning

    Output:
    - A `CompoundEntry` object:
        - `original_name`: the input name
        - `components`: a list of `Component` objects, one for each
plausible standardized interpretation

    Your explanation will be used for validation and traceability.
    """
    original_name: str = dspy.InputField(
        desc="One unresolved compound name that could not be resolved via
PubChem. May be a mixture, shorthand, or ambiguous term."
    )
    process_description: str = dspy.InputField(
        desc="The full description of the chemical process this name
appears in. Use this to infer chemical identity."
    )
    compound_entry: CompoundEntry = dspy.OutputField(
        desc="A mapping of the original ambiguous name to one or more
plausible standardized components."
    )
    explanation: str = dspy.OutputField(
        desc="Justification for your interpretation. Include how process
context, chemical reasoning, or known reaction products informed your
choices."
    )
```

Examine reactors for additional compounds:

```
class ReactionCompoundDiscovery(dspy.Signature):
    """
    You are an expert chemical process analyst specializing in industrial reactions.

    Your task is to analyze the provided process description together with a list of known chemicals associated exclusively with a **specific chemical reactor unit** (including all incoming reactants and possible products directly related to that unit).

    You must determine:
    - Whether there are any additional chemical compounds involved **only in the reaction occurring within the given reactor unit**, beyond those listed.
    - Explicitly suggest these additional compounds by name.

    You are given:
    - The complete text of the process description.
    - The reactor unit under consideration.
    - A unified list of chemicals relevant to this reactor unit (both incoming and potential products), which may include raw or standardized names.
    - A translation lookup dictionary to resolve ambiguous or unclear chemical names.

    Important:
    - Restrict your analysis strictly to the reaction chemistry happening inside the specified reactor unit.
    - Do not include or suggest chemicals that appear elsewhere in the overall process but are unrelated to this specific reactor.
    - Chemicals introduced after or before this reactor in the process
```

```
    flow that do not participate in the reaction at this unit must be
    excluded.
    - Use the process context to identify minor components, intermediates,
     byproducts, catalysts, or solvents that are very likely involved in
    this reactor's chemistry but missing from the input list.
    - HOWEVER, do not add chemical species that do not have a basis in the
     text description!
    - When uncertain about chemical names, prefer the clearer,
    standardized terms from the translation lookup.

    You must return:
    - A list of additional chemical names that appear to participate
    directly in the reaction at the given reactor unit but are missing from
     the input list.
    """
    process_description: str = dspy.InputField()
    reactor_unit: UnitWithConnections = dspy.InputField()
    chemicals: List[str] = dspy.InputField()
    translation_lookup: Dict[str, List[str]] = dspy.InputField()

    additional_chemicals: List[str] = dspy.OutputField(
        desc="Chemicals that participate in the reaction but are missing
    from the input list."
    )
```

Extract reaction conditions:

```
class ReactionConditionsExtraction(dspy.Signature):
    """
    You are a chemical process engineer analyzing reactor operating
    conditions.

    Your task is to extract the **reaction temperature** and **reaction
    pressure** for a given reactor unit, based on the process description.
```

```
6
7     Focus only on the specific reactor unit provided. Use its name, type,
      and description to find relevant statements. Look for phrases like:
8
9     - "The reactor operates at 300 C  and 10 bar."
10    - "Reaction occurs at 500 K and elevated pressure."
11    - "A pressure of 100 psi is maintained during reaction."
12
13    Return both the **numeric value** and the **unit as written in the
      text** (e.g., "K", " C ", "atm", "bar", "psi").
14
15    If the temperature or pressure is not mentioned, return None for both
      the value and unit.
16
17    Do not infer values unless clearly stated in relation to the given
      unit.
18
19    Return:
20    - Reaction temperature value and unit
21    - Reaction pressure value and unit
22    """
23    process_description: str = dspy.InputField()
24    reactor_unit: UnitWithConnections = dspy.InputField()
25
26    temperature_value: Union[None, float] = dspy.OutputField(desc="
      Reaction temperature value")
27    temperature_unit: Union[None, Literal[" C ", "K", " F "]] = dspy.
      OutputField(desc="Temperature unit from the text")
28    pressure_value: Union[None,float] = dspy.OutputField(desc="Reaction
      pressure value")
29    pressure_unit: Union[None, Literal["bar", "atm", "Pa", "kPa", "MPa", "
      psi", "psig", "psia"]] = dspy.OutputField(desc="Pressure unit from the
      text")
```

Extract pressure level from distillation-like separations:

```
class DistillationPressureExtraction(dspy.Signature):
    """
    You are a chemical process engineer analyzing the operating pressure conditions
    of a distillation column.

    Your task is to extract the **top (overhead) pressure** and **bottom pressure**
    for the given distillation column, based on the process description.

    Focus only on the specific distillation column provided. Use its name, type,
    and description to find relevant statements. Look for phrases like:

    - "The column overhead operates at 1.2 bar."
    - "Top pressure is maintained at 80 kPa."
    - "Bottoms pressure reaches 2 atm."
    - "The column operates under a condenser pressure of 3 bar."

    Return the **numeric value** and the **unit exactly as written in the text**
    (e.g., "bar", "atm", "Pa", "kPa", "MPa", "psi", "psig", "psia").

    If a specific pressure (top or bottom) is not mentioned, return None for both
    the value and the unit.

    If a pressure value is mentioned but it is unclear whether it refers to
    the top or the bottom of the column, return it for both **top and bottom pressure**.
```

```
25
26      Do not infer or assume pressure values unless clearly stated for this
        column.
27
28      Return:
29      - Top pressure value and unit
30      - Bottom pressure value and unit
31      """
32      process_description: str = dspy.InputField()
33      distillation_column: UnitWithConnections = dspy.InputField()
34
35      top_pressure_value: Union[None, float] = dspy.OutputField(desc="
        Overhead/Condenser pressure value")
36      top_pressure_unit: Union[None, Literal["bar", "atm", "Pa", "kPa", "MPa
        ", "psi", "psig", "psia"]] = dspy.OutputField(desc="Overhead pressure
        unit from text")
37
38      bottom_pressure_value: Union[None, float] = dspy.OutputField(desc="
        Bottoms/Reboiler pressure value (or ambiguous pressure)")
39      bottom_pressure_unit: Union[None, Literal["bar", "atm", "Pa", "kPa", "
        MPa", "psi", "psig", "psia"]] = dspy.OutputField(desc="Bottoms pressure
         unit from text")
```

Infer which components are recycled:

```
1  class RecycleChemicalInference(dspy.Signature):
2      """
3      You are a domain-aware assistant specializing in interpreting chemical
       engineering flowsheets.
4
5      Your task is to identify the **most likely recycled chemical(s)** at a
       specific unit operation where the flow path enters a recycle loop.
6
7      You are given:
```

```
    - The full description of the industrial process.
    - The current unit operation, provided as a structured object including its type and connections.
    - The name of the outgoing connection (e.g., 'top', 'bottom') that leads into the recycle path.
    - A list of all known chemicals involved anywhere in the process via `translation_lookup`.

    Guidelines:
    - Prefer a **single recycled chemical** when there is a clear candidate.
    - However, if it is chemically or operationally justified (e.g., azeotropes, unreacted feedstock, incomplete separations), return **multiple plausible recycled chemicals**.
    - Use the process description and unit structure to reason about typical recycle behavior.
    - Use `translation_lookup` to ensure chemical names are standardized.
    - If uncertain, return a small set of likely candidates rather than overgeneralizing.

    You must return:
    - A list of one or more recycled chemical names (as standardized strings).
    """
    process_description: str = dspy.InputField()
    unit: UnitWithConnections = dspy.InputField()
    connection_label: str = dspy.InputField()
    translation_lookup: Dict[str, List[str]] = dspy.InputField()

    recycled_chemicals: List[str] = dspy.OutputField(
        desc="List of one or more most likely recycled chemical names through this connection"
    )
```

Flag azeotropes:

```
class AzeotropeDetection(dspy.Signature):
    """
    You are a precise chemical process analyst specializing in phase equilibria and azeotrope identification.

    Your task is to determine whether an azeotrope is likely to exist in a specified distillation step, and if so,
    to identify which components from the given top and bottom product streams are involved.

    You are given:
    - The complete text of the process description.
    - A specific distillation-type unit operation, including its name and other available details.
    - A list of chemical components present in the top (overhead) product stream of the unit.
    - A list of chemical components present in the bottom (residue) product stream of the unit.

    Critical guidance:
    - **Base your reasoning primarily on explicit or implied information from the process description**.
    - Use chemical knowledge only to interpret and clarify what the process description suggests - do not override it.
    - Focus exclusively on the distillation unit provided.
    - Only consider azeotropes formed from the given top and bottom stream chemicals.
    - If no azeotrope is likely according to the process description, state so clearly.
    - If an azeotrope is likely, list all involved components and ensure they are drawn from the provided chemical lists.
```

```
21
22      You must return:
23      - Whether an azeotrope exists (True/False).
24      - The list of components (from top and/or bottom streams) involved in
    the azeotrope, if any.
25      """
26      process_description: str = dspy.InputField()
27      distillation_unit: UnitWithConnections = dspy.InputField()
28      top_chems: List[str] = dspy.InputField()
29      bottom_chems: List[str] = dspy.InputField()
30
31      azeotrope_exists: bool = dspy.OutputField(
32          desc="True if the process description indicates or strongly
    implies an azeotrope in this unit, otherwise False."
33      )
34      azeotrope_components: List[str] = dspy.OutputField(
35          desc="List of components (subset of top_chems and/or bottom_chems)
     involved in the azeotrope."
36      )
```

References

(1) Vogel, G.; Hirtreiter, E.; Schulze Balhorn, L.; Schweidtmann, A. M. SFILES 2.0: an extended text-based flowsheet representation. *Opt. Eng.* **2023**, *24*, 2911–2933.

(2) Morbach, J.; Yang, A.; Marquardt, W. OntoCAPE—A large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **2007**, *20*, 147–161.

(3) Kim, S.; Chen, J.; Cheng, T.; Gindulyte, A.; He, J.; He, S.; Li, Q.; Shoemaker, B. A.; Thiessen, P. A.; Yu, B.; Zaslavsky, L.; Zhang, J.; Bolton, E. E. PubChem in 2021: new data content and improved web interfaces. *Nucleic Acids Res.* **2021**, *49*, D1388–D1395.

(4) Landrum, G. RDKit: Open-source cheminformatics. 2010.

(5) Rehner, P.; Bauer, G.; Gross, J. FeOs: an open-source framework for equations of state and classical density functional theory. *Ind. Eng. Chem. Res.* **2023**, *62*, 5347–5357.

(6) Esper, T.; Bauer, G.; Rehner, P.; Gross, J. PCP-SAFT parameters of pure substances using large experimental databases. *Ind. Eng. Chem. Res.* **2023**, *62*, 15300–15310.

(7) Winter, B.; Rehner, P.; Esper, T.; Schilling, J.; Bardow, A. Understanding the language of molecules: predicting pure component parameters for the PC-SAFT equation of state from SMILES. *Digital Discovery* **2025**, *4*, 1142–1157.

(8) Commenge, J.-M.; Piña-Martinez, A. Data-driven initialization of evolutionary methods for process synthesis considering centrality and diversity criteria. *Comput. Chem. Eng.* **2026**, *204*, 109416.

(9) Neveux, T. Ab-initio process synthesis using evolutionary programming. *Chem. Eng. Sci.* **2018**, *185*, 209–221.

(10) Shervashidze, N.; Schweitzer, P.; van Leeuwen, E. J.; Mehlhorn, K.; Borgwardt, K. M. Weisfeiler-Lehman graph kernels. *J. Mach. Learn. Res.* **2011**, *12*, 2539–2561.

(11) Borgwardt, K.; Ghisu, E.; Llinares-López, F.; O'Bray, L.; Rieck, B. Graph Kernels: State-of-the-Art and future challenges. *Found. Trends Mach. Learn.* **2020**, *13*, 531–712.

(12) *Ullmann's Encyclopedia of Industrial Chemistry*; John Wiley & Sons, Ltd: Hoboken, USA.

(13) *Process Economics Program (PEP) Yearbook*; IHS Markit: London, UK.

(14) Rossberg, M.; Lendle, W.; Pfleiderer, G.; Tögel, A.; Torkelson, T. R.; Beutel, K. K. *Ullmann's Encyclopedia of Industrial Chemistry*; John Wiley & Sons, Ltd, Hoboken, USA, 2011.

(15) Kahl, T.; Schröder, K.-W.; Lawrence, F. R.; Marshall, W. J.; Höke, H.; Jäckh, R. *Ullmann's Encyclopedia of Industrial Chemistry*; John Wiley & Sons, Ltd, Hoboken, USA, 2011.

(16) Le Berre, C.; Serp, P.; Kalck, P.; Torrence, G. P. *Ullmann's Encyclopedia of Industrial Chemistry*; John Wiley & Sons, Ltd, Hoboken, USA, 2014.