Digital Discovery

ROYAL SOCIETY OF CHEMISTRY

PAPER

View Article Online
View Journal | View Issue

Cite this: *Digital Discovery*, 2025, **4**, 1534

Natural language processing for automated workflow and knowledge graph generation in self-driving labs†

Bastian Ruehle *

Natural language processing with the help of large language models such as ChatGPT has become ubiquitous in many software applications and allows users to interact even with complex hardware or software in an intuitive way. The recent concepts of Self-Driving Labs and Material Acceleration Platforms stand to benefit greatly from making them more accessible to a broader scientific community through enhanced user-friendliness or even completely automated ways of generating experimental workflows that can be run on the complex hardware of the platform from user input or previously published procedures. Here, two new datasets with over 1.5 million experimental procedures and their (semi)automatic annotations as action graphs, *i.e.*, structured output, were created and used for training two different transformer-based large language models. These models strike a balance between performance, generality, and fitness for purpose and can be hosted and run on standard consumer-grade hardware. Furthermore, the generation of node graphs from these action graphs as a user-friendly and intuitive way of visualizing and modifying synthesis workflows that can be run on the hardware of a Self-Driving Lab or Material Acceleration Platform is explored. Lastly, it is discussed how knowledge graphs – following an ontology imposed by the underlying node setup and software architecture – can be generated from the node graphs. All resources, including the datasets, the fully trained large language models, the node editor, and scripts for querying and visualizing the knowledge graphs are made publicly available.

Received 13th February 2025
Accepted 2nd May 2025

DOI: 10.1039/d5dd00063g

rsc.li/digitaldiscovery

Introduction

Self-driving labs (SDLs) and materials acceleration platforms (MAPs) have received much attention in recent years.[1,2] Automating the synthesis of new molecules and materials on autonomous robotic platforms that integrate artificial intelligence (AI) and machine learning (ML) tools in a closed loop process has great potential for accelerating or even completely changing the way chemical experiments will be conducted in the future.[3] However, to make these techniques viable, appealing, and approachable for a wide variety of primarily synthetically trained chemists and material scientists, it is important to ensure that the entry barriers to using these new tools are as low as possible. Hence, an intuitive way for generating automated synthesis workflows on such hardware using suitable interfaces to the robotic platforms will be a key component for the broad application of MAPs and SDLs in the scientific community.

Natural language is perhaps one of the most intuitive ways for humans to express their intents and interact with any given system. The recent rise and large success of AI-based large language models (LLMs) demonstrates this very well and paved the way for numerous applications that rely heavily on user interaction through natural language. Moreover, in chemistry – like in almost all scientific disciplines – the vast majority of the knowledge that has been generated over the past centuries is only available in the form of unstructured natural language in books or scientific publications rather than in structured, machine-readable and readily interoperable data.

For these reasons, natural language processing (NLP) is an obvious choice for generating automated synthesis workflows on robotic platforms such as SDLs and MAPs, either directly from user input, or from previously published experimental procedures. There are several examples, some as early as 2011,[4] in which rule-based algorithms, part-of-speech (POS) tagging, and named entity recognition (NER) are used to create structured output from unstructured experimental procedures.[5–13] While POS and NER can also be done using neural network architectures like convolutional neural networks (CNN) and recurrent neural networks (RNN) such as long-short-term-

Federal Institute for Materials Research and Testing (BAM), Richard-Willstaetter-Strasse 11, D-12489 Berlin, Germany. E-mail: bastian.ruehle@bam.de

† Electronic supplementary information (ESI) available: Links to further resources, detailed description of dataset creation steps, and further figures and tables. See DOI: https://doi.org/10.1039/d5dd00063g

memory (LSTM) or gated recurrent unit (GRU) networks instead of purely rule-based approaches,[10,11] the introduction of the transformer architecture[14] and with it the advent of LLMs such as the GPT model family[15] revolutionized the task of NLP. In general, transformer-based LLMs are more robust against grammatical or typographical errors than rule-based approaches and make the challenge of formulating precise yet generic rules that encompass all possible ways language can be used to convey the same meaning obsolete due to their high dimensional latent embeddings and their attention mechanisms.

Consequently, most recent efforts of integrating NLP with MAPs or SDLs are focused on using LLMs. Following the approach of generating action graphs from experimental procedures that can be executed on robotic hardware,[13] Vaucher *et al.* demonstrated the usefulness of LLMs for this task with a custom encoder–decoder transformer model with eight attention heads[16] that can generate executable actions for a proprietary hardware backend.[17] Yoshikawa *et al.*[18] used automated iterative prompting between a generator that queries proprietary OpenAI LLMs and a rule-based verifier for generating executable XDL code[13] on robotic hardware in a MAP.

In both cases, the weights of the fully trained neural networks are proprietary and not publicly available, and the networks are hosted on third party servers. However, there are certainly cases in which it is beneficial to have full control over the data in one's own IT infrastructure, for example when considering especially sensitive data, intellectual property data, or when running the software on PC systems that are not or cannot be connected to the internet to query online resources for security reasons. Moreover, making the fully trained LLMs publicly available gives other researchers the possibility to fine-tune the models according to their specific needs or domain-specific language.

With the recently published Llama series of LLMs,[19] very large, fully trained networks are made publicly available, however, these are prohibitively memory-consuming and slow when attempting to run them on standard, consumer-grade hardware typically used in edge computing and commonly used laboratory IT infrastructure. Hence, there is a need for fully open, publicly available LLMs that strike a balance between performance, generality, and fitness for purpose for the task of automatically generating workflows and executable code from natural language inputs.

For this reason, the fine-tuning and comparison of two different, pre-trained encoder–decoder transformer "surrogate" LLMs for the task of generating action graphs from experimental procedures written in unstructured, natural language are investigated and described in detail here. The training data for these models was compiled from over 1.5 million publicly available experimental procedures from the patent literature and annotated using a rule-based approach, as well as more general, but also much larger LLMs. The performance of the fully trained surrogate models is evaluated and compared using different metrics, and the fully trained models are made freely publicly available. Additionally, the ability of the surrogate models to adapt to experimental procedures from the domains of materials science, organic chemistry, inorganic chemistry, and patents that were not part of the training or evaluation dataset are discussed.

The action graphs generated by these models follow a simple markup language, *i.e.*, they represent structured output with a clear vocabulary and syntactic rules. In principle, such intermediate output can readily be turned into executable code by either a rule-based custom "compiler" or by another LLM. Here, the first approach is followed, albeit with an intermediate step. Rather than turning the action graphs generated from the natural language input directly into executable python code, a node setup in a node editor of a graphical user interface is generated first. Such node editors are often used for creating or representing workflows in a more accessible or "visual" way compared to classical, text-based programming. Representing the action graphs this way makes it easier for users without in-depth programming knowledge to modify or adjust the automatically generated action graphs in an intuitive way. Moreover, the node editor can also be used in a stand-alone way to build entirely new workflows from scratch.

Finally, the nature of the node-based setup with different input and output fields that are grouped hierarchically under one node, in combination with the underlying inheritance structure of the entities corresponding to nodes and input and output fields in an object-oriented programming language and the software architecture of the SDL backend, lend themselves excellently to the generation of knowledge graphs from these intermediate node graphs, which is also investigated in this work. It should be noted that these automatically generated knowledge graphs follow an ontology that is (currently) imposed by the underlying architecture of the software backend of the SDL. However, in the future, a common underlying ontology used for this type of representation could help to further harmonize and enhance sharing synthesis workflows even across different MAP or SDL platforms.

Results and discussion

Datasets

The datasets that were used for training the NLP models are based on the "Chemical reactions from US patents (1976-Sep2016)" dataset.[20] This dataset contains experimental descriptions that were automatically extracted from publicly available US patents and patent applications in the given period. The style of writing is very similar to the style of experimental procedures published in the experimental parts of scientific articles, especially in organic chemistry journals.

In a first approach, the dataset was annotated using ChemicalTagger,[4] which uses a rule-based part-of-speech (POS) tagging algorithm. The POS-tagged experimental procedures were further processed into action graphs using a python script that traverses the xml files generated by ChemicalTagger, parses them for *ActionPhrases*, *Molecules*, and their associated *Quantities*, and combines them in a (linear) action graph. Procedures with only one *ActionPhrase* (usually ⟨SYNTHESIZE⟩ or ⟨YIELD⟩) were excluded. This resulted in an annotated dataset with 1′573′734 entries consisting of experimental procedures and the

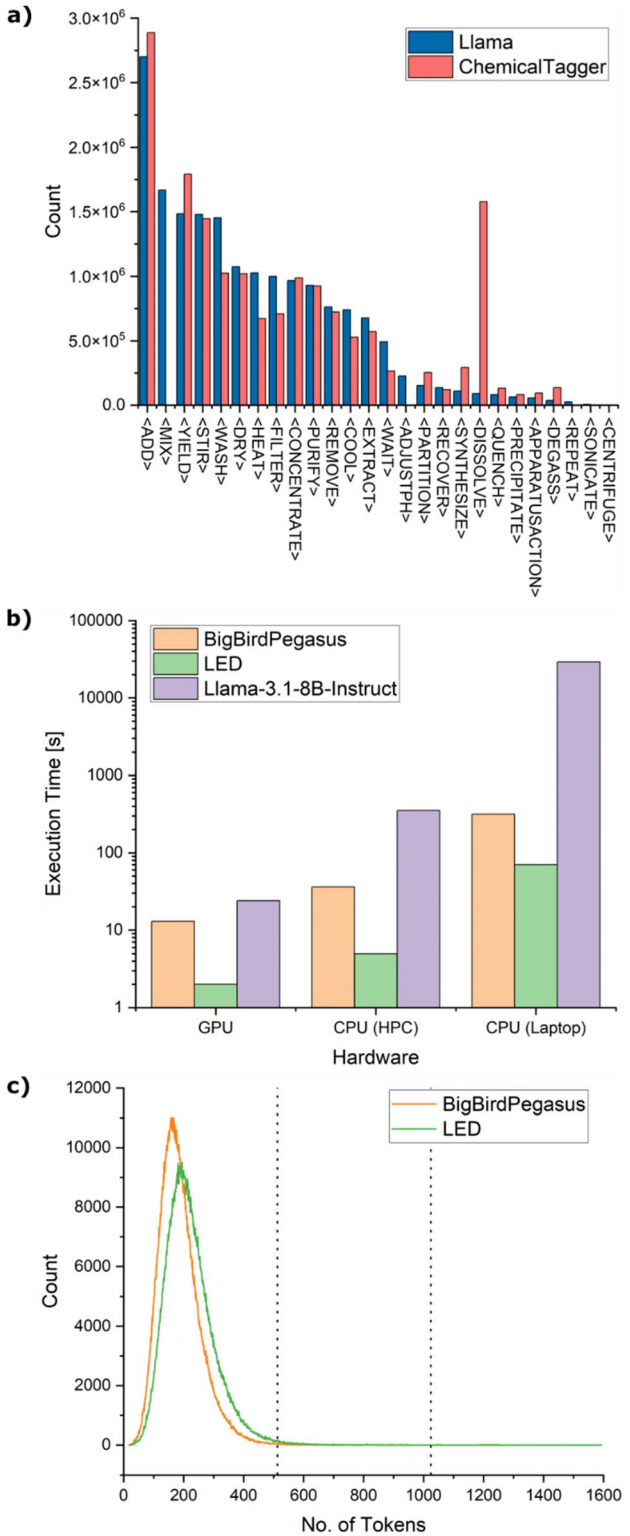

Fig. 1 (a) Occurrence of the action tags in the cleaned datasets annotated with ChemicalTagger and the Llama model. Some tags are only present in the dataset created by the Llama model. (b) Mean execution times for the different models on a GPU (A100-80 GB), a high-end CPU in a high-performance computing (HPC) cluster (Intel(R) Xeon(R) Gold 6342 CPU @ 2.80 GHz), and a CPU in a standard office laptop (Intel(R) Core(TM) i5-1235U @ 2.50 GHz). The execution time is displayed on a logarithmic scale. (c) Distribution of the number of tokens in the experimental procedures when tokenized with the pre-trained BigBirdPegasus and LED tokenizers. The cut-offs at 512 and 1024 tokens used during fine-tuning are indicated with a dashed line.

corresponding action graphs, named dataset_chemtagger_raw. The dataset was further cleaned by replacing commonly occurring non-ASCII characters such as μ or × with u and x, removing extra line-breaks, leading and trailing spaces, and grouping actions in cases where the same action tags occurred consecutively, giving a new dataset (named dataset_chemtagger_cleaned) which was later used for training the NLP surrogate models. In this dataset, the 21 action tags used by ChemicalTagger occurred with varying frequencies, ranging from 2′887′135 for the most common action ⟨ADD⟩ to 81′244 for the least common action ⟨PRECIPITATE⟩ (see Fig. 1a).

In a second approach, the recently published Llama-3.1-8B-Instruct[19] model was used for creating the annotated training data using in-context learning (ICL). The Llama v3.1 community license explicitly allows to use the outputs of the model for synthetic data generation and distillation. While the model weights are made publicly available and can be downloaded and used after a free sign-up process, hosting the model offline and using it for inference is still very resource intensive. Even the smallest model of the family with 8 billion parameters requires at least 16 GB of GPU RAM in its highest precision (fp16) just for the model weights, plus the memory that is needed for keeping the keys and values of each token in memory to leverage the models long available context length. This already exceeds the memory of most consumer-grade GPUs currently on the market. Moreover, the average inference time for creating an action graph of a single experimental procedure is still long, 24 seconds when run on a high-end GPU (A100-80GB), 352 seconds when run on a high-end CPU, or 29′109 seconds (8 hours) when run on a standard office laptop, as compared to 13, 36 and 316 seconds for a BigBirdPegasus-large model and 2, 5, and 70 seconds for a LongformerEncoderDecoder-base model (see Fig. 1b). Additionally, while the generated action graphs were generally of good quality, the model used its own action tags in 194′073 annotations (12.3%), even when being clearly instructed as part of its ICL prompt to only use tags from a pre-defined list of action tags (see ESI† for details). Overall, 2′956 different tags were used by the model in the annotations, instead of the 23 tags the model was instructed to use. Having such a large number of (unexpected) tags would make the generated action graphs very specific and also render the automatic downstream processing of action graphs much more complicated. This further corroborates the efforts for training and using a much smaller, domain-specific surrogate model for the specific task of action graph generation, rather than querying a more general, larger LLM using ICL.

More details about the (semi-)automatic annotation of the dataset with the help of the Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct models as well as manually performed substitutions for data cleanup can be found in the ESI.†

The final dataset (dataset_llama_cleaned) that was used for training the surrogate models contained 26 different tags (the

"original" 21 ChemicalTagger action tags, plus the newly added tags ⟨MIX⟩, ⟨ADJUSTPH⟩, ⟨REPEAT⟩, ⟨SONICATE⟩, and ⟨CENTRIFUGE⟩), occurring with frequencies varying between 2′700′427 for the most common action ⟨ADD⟩ to 1′453 for the least common action ⟨CENTRIFUGE⟩ (see Fig. 1a).

Models

Two different surrogate models were trained on the two cleaned datasets, namely a BigBirdPegasus model[21] and a LongformerEncoderDecoder (LED) model.[22] Both models have been used in the past for sequence-to-sequence language modelling tasks (such as machine translation or text summarization) and shown excellent performance.

Both are transformer models based on the BART architecture with a BERT-like encoder and a GPT-like decoder but handle the attention mechanism differently to circumvent the quadratic dependency on sequence length, *i.e.*, by using block-sparse attention or local attention to tokens within a certain window-size, respectively.

Pre-trained versions of both models and tokenizers were used for the fine-tuning approach (see the Methods section for details), but the action tokens were added to the respective pre-trained tokenizers. After tokenization of the datasets, the mean length of the input sequences was 187 tokens when using the BigBirdPegasus tokenizer and 221 tokens when using the LED tokenizer (see Fig. 1c). Given the length distributions of the tokenized datasets, a maximum sequence length of 512 for training the BigBirdPegasus model was used (longer sequences were truncated). Hence, the original full attention instead of BigBird's block-sparse attention is used since there is no benefit in using block-sparse attention for sequence lengths less than 1024 tokens. For the LED model, a maximum sequence length of 1024 is used, since the window size of the pre-trained model was set to 1024, and the sequences have to be padded to a multiple of the window size. To keep the total training time similar for both models despite the longer sequence length used for the LED model, the "base" version of the LED model is used, as compared to the "large" version of the BigBirdPegasus model. The models were then trained individually on the training split of the datasets (see Methods section for training hyperparameters and details) and evaluated on the evaluation split.

Evaluation metrics

After training was completed, the surrogate models were evaluated on the evaluation split of the dataset using the ROUGE and BLEU scores. Both are commonly used metrics that range from 0 to 1 (higher is better) to assess the quality of text generated by NLP models, *e.g.*, in machine translations or text summarization. The action graphs generated by Llama-3.1-8B-Instruct for the validation split were used as ground truth labels. The results are shown in Table 1.

As already stated for the "Chemical reactions from US patents (1976-Sep2016)" dataset, duplicate reactions are frequent in the dataset due to the same or highly similar text occurring in multiple patents, and many reactions from patent applications also appear later in granted patents. This means that no strict separation of training and validation data can be guaranteed, and the metrics reported here for the validation dataset might be artificially inflated. This is however hard to avoid in practice, and the metrics are only used here for a relative comparison of the models trained in this work without attempting a comparison with the scores reported by other authors.

It should also be noted that the ChemicalTagger dataset contained 5 tags less than the Llama dataset, including the very frequently used tag ⟨MIX⟩. Since the evaluation split of the Llama dataset was used as ground truth for calculating the scores, the maximum achievable score for the surrogate models trained on the ChemicalTagger dataset is less than 1.0. Hence, for a better comparison, the scores the ground truth labels from the validation split of the ChemicalTagger dataset achieved when compared to the ground truth labels of the Llama dataset are also given (rows 2 and 3 in Table 1). Interestingly, the surrogate models trained on the ChemicalTagger datasets achieved very similar (and in some cases even slightly higher) scores than the validation split of the ChemicalTagger dataset itself, indicating the ability of the models to leverage the context-sensitive token embeddings they learned during the pre-training phase to match (or even outperform) rule-based annotations. Lastly, these data also reveal that the raw output of these specialized surrogate models comes close to the raw output of the more general and much larger LLM (Llama_raw) while using only a fraction of the time and computational resources and not requiring any ICL prompt engineering, post-processing, or iterative prompting to remove unwanted action

Table 1 Evaluation metrics of the surrogate models trained on the two different datasets when compared to the cleaned dataset created by the Llama models as the ground truth. The results for the raw and cleaned datasets created by ChemicalTagger and the raw dataset created by Llama-3.1-8B-Instruct are also given for comparison

Model or dataset	ROUGE1	ROUGE2	ROUGEL	ROUGELsum	BLEU
Llama_raw	0.9777	0.9641	0.9759	0.9759	0.9366
ChemicalTagger_raw	0.7850	0.6165	0.7129	0.7129	0.5817
ChemicalTagger_cleaned	0.7895	0.6229	0.7163	0.7163	0.5835
BigBirdPegasus_Llama	0.9483	0.9046	0.9350	0.9350	0.8781
LED-Base-16384_Llama	0.9529	0.9126	0.9402	0.9402	0.9096
BigBirdPegasus_ChemicalTagger	0.7883	0.6208	0.7153	0.7153	0.5649
LED-Base-16384_ChemicalTagger	0.7898	0.6232	0.7166	0.7166	0.5841

tags, syntax errors, or other "hallucinated" output often generated by the Llama models (see also ESI† for more details on required data clean-up steps for the output generated by Llama models).

Example outputs for experimental procedures from the domains of materials science, organic chemistry, inorganic chemistry, and a patent that were not part of the training or evaluation dataset are given in the ESI (Table S3),† demonstrating the ability of the models to also adapt to procedures from other domains. In general, the models trained on the Llama dataset produced better outputs and struggled less with inconsistent uses of spaces or other characters in the example procedures.

For example, in the part of an experimental procedure that read "A mixture of […] and K 2 CO 3 (138 g, 1.0 mol) in DMF (500 mL) was heated at 95 °C" (Example 5 in Table S3†), both models trained on the ChemicalTagger dataset failed to annotate the addition of K_2CO_3 (as well as the subsequent addition of DMF) correctly, while the models trained on the Llama datasets correctly annotated this step. This is clearly a limitation of the training dataset rather than the models themselves, perhaps such examples were also incorrectly annotated by ChemicalTagger or the parser script extracting the action graphs from the ChemicalTagger xml files.

Another noteworthy example was that the models trained on the ChemicalTagger dataset used ⟨RECOVER⟩ for the centrifugation step in the procedure that read "The nanoparticles were collected by centrifugation (10 min at 7197 rcf) […]" (Example 1 in Table S3†), since there is no ⟨CENTRIFUGE⟩ tag in the training data annotated with ChemicalTagger. Interestingly, the models trained on the Llama datasets, that included the ⟨CENTRIFUGE⟩ tag, did not annotate this step at all.

Lastly, it should be highlighted that especially the BigBird-Pegasus model trained on the Llama dataset was capable of some remarkable generalizations that were most likely never presented to the model during the fine-tuning phase.

In several (but not all) cases, it could detect repeated actions and added the corresponding actions several times. For example, in the procedure that read "After washing the precipitate with cold water (2 × 25 mL) […]" (Example 6 in Table S3†), it was the only model that created the annotation "⟨WASH⟩ cold water 25 mL cold water 25 mL", with the other models annotating this action as "⟨WASH⟩ cold water 2 × 25 mL". Likewise, in the procedure that read "[…] washed 2× with water (2 × 90 mL), 2× with ethanol (2 × 90 mL) and 2× with toluene (2 × 90 mL) […]" (Example 1 in Table S3†), it created the annotation "⟨WASH⟩ water 2 × 90 mL water 2 × 90 mL ethanol 2 × 90 mL ethanol 2 × 90 mL toluene 2 × 90 mL", *i.e.*, correctly repeating the washing steps with water and ethanol twice. Strangely it did not do the same for toluene though. While the correct meaning of repeating the action could perhaps also be deduced in a post-processing step from the fact that the volumes in the annotations created by the other models have a "2×" modifier, it is interesting that the BigBirdPegasus model was the only one to create two separate actions here.

Even more remarkable is the annotation it produced for the part of the procedure that read "[X] and [Y] were suspended in a 1 : 1 mixture of water and *tert*-butyl alcohol (12 mL)" (Example 6 in Table S3†). Here, it produced the annotation "⟨ADD⟩ [X] [Y] water 6 mL *tert*-butyl alcohol 6 mL", *i.e.*, correctly reasoning that 12 mL of a 1 : 1 mixture of water and *tert*-butyl alcohol implies adding 6 mL of water and 6 mL of *tert*-butyl alcohol.

From action graphs to node graphs

The action graphs that are generated by the fine-tuned LLM models represent a very simple, structured markup language. They are very generic in nature, and not tailored or restricted to specific hardware interfaces, APIs, or programming languages. However, due to the structured nature of the output with a clearly defined vocabulary and syntactic rules, these action graphs can be readily converted into other formats, such as XDL[13,18] or – as demonstrated here – a node graph, by using standard rule-based text parsing, similar to what a compiler would do. These can then be further translated into executable code (*e.g.*, python scripts) that can be run on the specific hardware of a self-driving lab, as demonstrated below.

Node graphs that can be created and edited in node editors are a popular and accessible way of representing workflows with dynamic inputs and outputs and are often used for "visual programming" in fields such as graphics design, video editing, games design, and machine learning. They group functional subunits (such as classes and methods) into nodes, whose output can be used as the input for other nodes by simply connecting the two nodes together. While arguably more restricted and less flexible than writing code in "traditional", text-based programming languages, the visual representation and simplicity of connecting nodes with lines makes them generally easier to use and more accessible for people who are not familiar with programming languages. Especially in use cases in which the underlying workflows are typically very similar and the use is limited by other external constraints, such as the availability of hardware modules in an SDL, the somewhat limited flexibility of the node graphs is usually an acceptable trade-off. These characteristics make them very interesting as part of a graphical user interface (GUI) for setting up reactions on an SDL, without requiring the user to know any "traditional" text-based programming language, the underlying hardware components, or even the API used for communicating with the hardware components.

To demonstrate the usefulness of this approach, a node graph editor for our recently published MAP for nano- and advanced materials syntheses MINERVA[23] is implemented. The inputs and outputs of the nodes are automatically generated by inspecting the signatures of the constructors and methods of the underlying subunits of the API that are represented by the nodes. This makes code maintenance easier, since any changes to the call signatures and any additional parameters that might be added to methods in the future will be automatically incorporated into the nodes as inputs. Additionally, it helps with user-friendliness and documentation by directly using the text from the docstrings of the classes and methods as tooltips for the individual node inputs. Lastly, it makes translating the node graph back to executable python code that can be directly run on the SDL straightforward.

To construct an executable node graph from an action graph, the generic action tags used by the action graph have to be mapped to operations that can be performed by the hardware of the respective MAP or SDL. For our MAP, the action tags are mapped to the high-level methods defined by our API, namely 'Add Chemical', 'Heat', 'Infuse While Heating', 'Remove Supernatant and Redisperse', 'Sonicate', 'Centrifuge', and 'Transfer Content to Container' (see ref. 23 for details). Here, certain domain-specific assumptions were made to take into account that our MAP is designed for nano and advanced materials synthesis and also to accommodate the resulting limitations imposed by the available hardware modules of our MAP. For example, the action tags ⟨EXTRACT⟩, ⟨PARTITION⟩, ⟨REMOVE⟩, ⟨CONCENTRATE⟩, ⟨FILTER⟩, ⟨DRY⟩, ⟨CENTRIFUGE⟩, and ⟨RECOVER⟩ are all simply mapped to the 'Centrifuge' node, followed by a 'Transfer Content to Container' node for transferring the supernatant to a waste container. In general, this will not be a valid approach, since in organic chemistry, ⟨EXTRACT⟩ and ⟨PARTITION⟩ usually refer to a liquid–liquid extraction or partitioning a substance between two immiscible liquids, ⟨REMOVE⟩ and ⟨CONCENTRATE⟩ usually refer to the rotary evaporation of a solvent, and for a ⟨FILTER⟩ step, it has to be deduced whether the solids or the filtrate contain the compound of interest. However, in nano- and advanced materials synthesis, it is usually safe to assume that the user is interested in the solid parts (*i.e.*, the nanomaterials), and that the solids can be separated from the solution *via* centrifugation, followed by removing the supernatant. Moreover, in our MAP, centrifugation is currently the only available method for separating or extracting a solid material from a synthesis solution. Following the same reasoning, ⟨PURIFY⟩ and ⟨WASH⟩ are represented by a 'Centrifuge' node followed by 'Remove Supernatant and Redisperse' node, ⟨DISSOLVE⟩ is represented by an 'Add' node followed by a 'Heat' node for stirring, ⟨ADD⟩, ⟨PRECIPITATE⟩, and ⟨QUENCH ⟩ are all represented by an 'Add' node, a ⟨HEAT⟩ tag followed by an ⟨ADD⟩ tag is represented by an 'Infuse While Heating' node, and a single ⟨HEAT⟩, ⟨COOL⟩, ⟨MIX⟩ and ⟨STIR⟩ action is represented by a 'Heat' node, potentially using a stirring speed of 0 rpm for a simple heating step or 25 °C (room temperature) as the heating temperature for a simple stirring step, depending on the parameters provided under this action tag.

As mentioned above, the exact implementation of representing the action graph as a node graph is highly specific to the domain and the underlying hardware and software platform of the MAP. The action tokens in the training datasets of the NLP models, and hence the generated action graphs, were intentionally kept generic to make the action graphs less domain and hardware specific and more broadly applicable in different scientific domains, and this substitution is only done when generating the node graphs specific for the MAP.

During creation of node graphs from action graphs, certain heuristics and error correction measures are implemented as well to handle imprecise experimental descriptions or missing parameters from the synthesis protocol and the user is informed of the measures taken through a warning message. For example, imprecise time (*e.g.*, overnight), temperature (room temperature), or stirring speed (vigorous stirring) specifications can be replaced with default values, such as 16 hours, 25 °C and 600 rpm, while missing process parameters can be substituted with pre-defined default values (*e.g.*, always using 8000 rpm if no centrifugation speed is specified). These substitutions and pre-defined default values are often also domain-specific, and more details for the current implementation can be found in the ESI.†

These MAP-specific node graphs can then be further processed to generate executable code. In our case, this step is rather straightforward since the nodes are directly generated from the high-level methods of the python API controlling our MAP, but in principle, they can also be used to generate *e.g.*, XDL code that can then be executed. It should be noted that in general, some user input is still required before converting the node graph to executable code. Besides cleaning up any incorrect actions, this typically also involves specifying the positions of the required starting materials and reagents on the platform, as well as the reaction vessels that should be used during the reaction. For commonly used chemicals (such as solvents) and "fixed" containers (such as waste containers) that are always connected to the same ports of a valve, these can of course also be hard-coded into the node generation process and be saved as part of the hardware configuration of the MAP. The same applies to cases in which always the same chemicals are used for the reactions run on the MAP, *e.g.*, during one synthesis campaign. Since our MAP is very material-agnostic[23] and is generally used in a research-centered environment with constantly changing reactants, reagents, and target materials, this step is typically left to the user in our MAP. A full example of this processing pipeline from natural language input to an action graph (Example 2 in Table S3, ESI†), a node graph

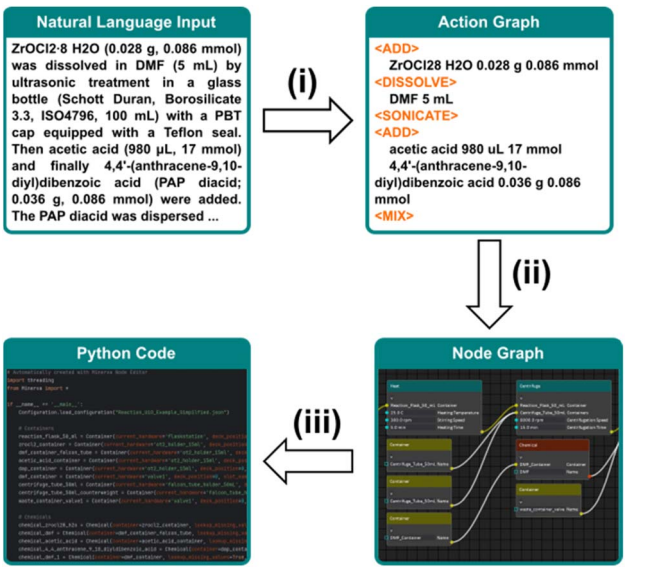

Fig. 2 Schematic representation of the pipeline for generating (i) an action graph, (ii) a node graph, and (iii) executable Python code from natural language input.

(Fig. S4, ESI†), and executable python code (Table S4, ESI†) is given schematically in Fig. 2 and in more detail in the ESI.†

From node graphs to knowledge-graphs (and ontologies)

Using the class-based hierarchy and inheritance structure of the classes and methods underlying the nodes, their input and output field types, as well as their connections in the node graphs, an application- or domain-level ontology and knowledge graph can be automatically deduced from the node graphs as well, besides executable code. Again, the exact implementation will be very domain specific, but in our case, all connections between nodes represent a "is_used_by" respectively "uses_material_from" relationship. The only exception is the connection between 'Container' nodes and 'Chemical' nodes that represent a "contains_chemical" respectively "is_contained_in" relation. The node input parameters all represent a "has_property" relationship, while the type of the input variable associated with this field (*e.g.*, a float, bool, Container, Chemical, Volume, *etc.*) represents a "is_a" relationship.

By following the inheritance structure of the individual classes and their method resolution order (MRO), the graph can be expanded down to the very basic (abstract) base classes. For example, in our API, the "Volume" class inherits from the abstract base class "Quantity" (*i.e.*, "Volume"-"is_a"-"Quantity"), and requires the two inputs – "value" and "unit" – in its constructor (*i.e.*, "Volume"-"has_property"-"value" and "Volume"-"has_property"-"unit"). The value itself is of type float (*i.e.*, "value"-"is_a"-"float"), the unit is of type string (*i.e.*, "unit"-"is_a"-"string"). Ultimately, both floats and strings (as well as all other custom or built-in classes and types in Python) are

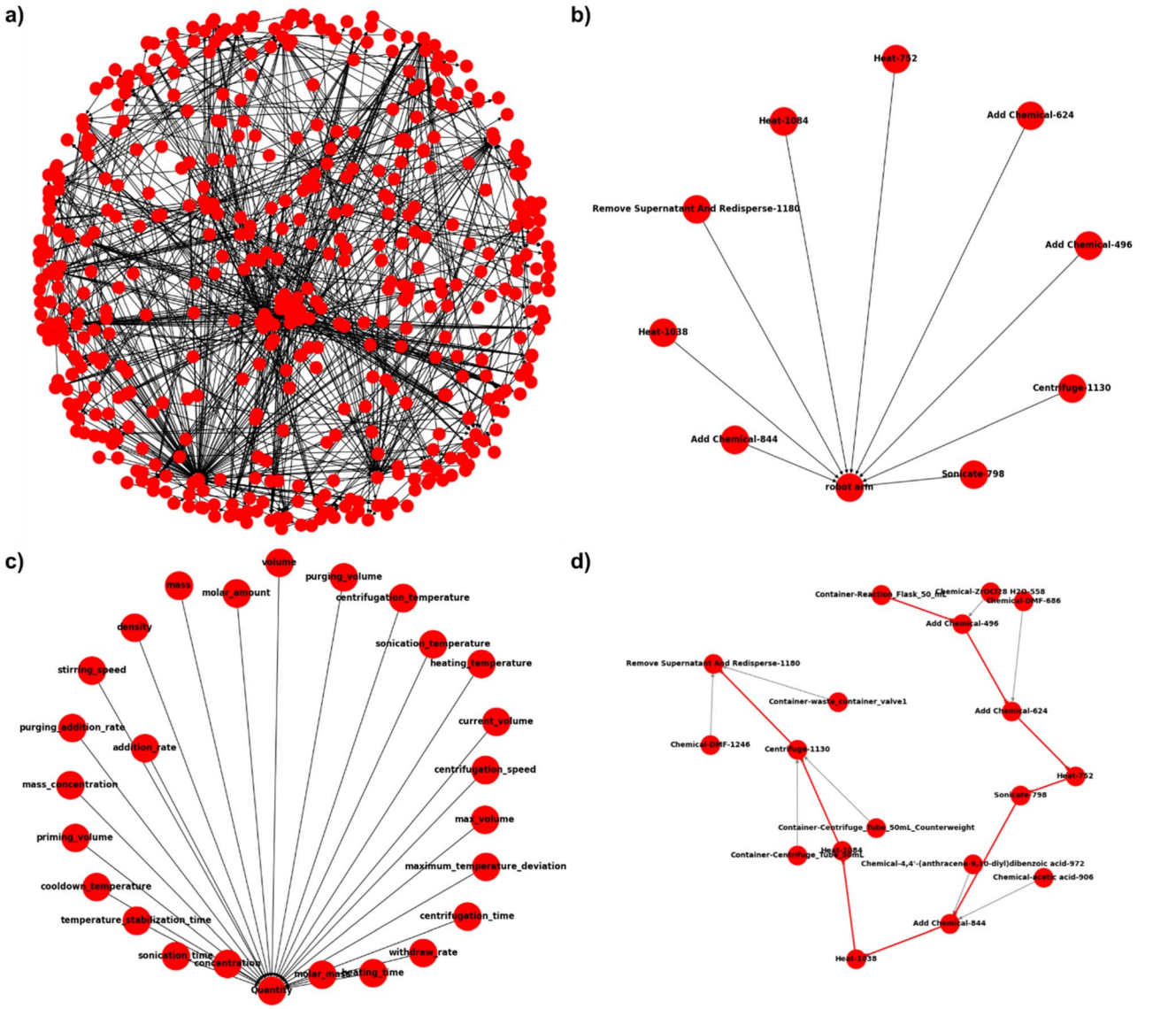

Fig. 3 (a) Full knowledge graph obtained from the node graph shown in Fig. S1† with all entities and their connections. (b) Example query ('? – has_property – robot arm') of the knowledge graph. (c) Example query ('? – is_a – Quantity') of the knowledge graph. (c) Example query ('Shortest Path from Flask Node to Remove_Supernatant_and_Redisperse Node') of the knowledge graph.

"objects" (this relationship is excluded from the graph since it carries no added information in this context).

Lastly, the fields have values associated with them represented by a "has_value" relationship (*e.g.*, for a volume of 5 mL: "value"-"has_value"-"5" and "unit"-"has_value"-"mL"). This way, a knowledge graph can be constructed from the underlying ontology. Fig. 3 shows an example of a full knowledge graph as well as three example queries of this graph that were generated from the node graph shown in Fig. S1 in the ESI.†

Methods

Dataset generation

The datasets for surrogate model training were generated from the "Chemical reactions from US patents (1976-Sep2016)" dataset[20] by extracting the experimental procedures using a python script. A first dataset was created by rule-based part-of-speech (POS) tagging with ChemicalTagger,[4] followed by exporting the results as xml, and constructing the desired action graph using a rule-based parsing of the POS-tagged xml file. A second dataset was created by annotating the same experimental procedures with the Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct models[19] using in-context learning (ICL), as well as some manual clean-up instead of the rule-based approach outlined above (see ESI† for details). Both datasets were further post-processed to clean up artifacts as explained in the ESI† and then used for model training and validation.

Benchmarking

For comparing execution times on a high-end GPU (A100, 80 GB), a single, high-end CPU in an HPC cluster (Intel(R) Xeon(R) Gold 6342 CPU @ 2.80 GHz, using 12/24 logical cores and no restrictions on RAM (750 GB total available)), and a standard office laptop CPU (12th Gen Intel(R) Core(TM) i5-1235U @ 2.50 GHz, using 12/12 logical cores and 16 GB of RAM), inference was run three times for three different inputs with raw input lengths before tokenization of 508, 650, and 860 characters (chosen from the 25th, 50th, and 75th percentile of the dataset), and the execution times were averaged. For the surrogate models, the models fine-tuned on the ChemicalTagger dataset were used. For the Llama-3.1-8B-Instruct model, ICL prompt template 1 (see ESI†) was used. No batching was applied. The time does not include loading the models and tokenizers or any pre- or post-processing, but it includes creating the pipeline and running tokenization and inference.

LLM finetuning

As the starting point of the BigBirdPegasus model,[21] a checkpoint of Google's BigBirdPegasusForConditionalGeneration after finetuning for summarization on the big_patent dataset[24] is used. As the starting point for the Longformer Encoder Decoder (LED) model,[22] the pretrained led-base-16384 model from AllenAI[25] is employed. The action tokens are added to the tokenizer, the generated dataset is randomly split into training and validation sets (90 : 10) and the models are trained on A100-80 GB GPUs for 885′225 steps (5 epochs) on the training split, using a batch size of 8, an initial learning rate of 5×10^{-5} with a 0.05 warmup ratio, and a cosine weight decay. All other hyperparameters used the default values.

Conclusions

Two new, large, (semi-)automatically annotated datasets were created from experimental procedures that were extracted from patents and patent applications. These datasets were used to train surrogate LLMs on the task of creating action graphs, *i.e.*, simple structured output, from the unstructured input. The LLMs that were trained, a BigBirdPegasus and a Longformer Encoder Decoder model, strike a good balance between performance, generality, and fitness for purpose and can be hosted and run on standard consumer-grade hardware. The generated action graphs were generally of high quality, with some remarkable generalizations especially from the BigBird-Pegasus model. Moreover, the training dataset (and hence the trained models) were kept rather broad in scope and their applicability to experimental procedures from different domains was investigated, namely materials science, organic chemistry, inorganic chemistry, and patent literature.

Further, the possibility of using a node editor for creating, visualizing, and modifying synthesis workflows for MAPs and SDLs is presented. The node graphs can either be created automatically from the action graphs generated by the LLMs or from scratch by the user. They offer an intuitive way of creating automated synthesis workflows that can be run on robotic hardware without requiring classical programming skills.

By dynamically creating the nodes in the node editor from the API of the software backend underlying the MAP or SDL, it is rather straightforward to generate executable code from these node graphs that can then be used to directly run the planned experiment(s) on the hardware platform. Moreover, the nature of the node graph itself as well as the fact that the nodes are dynamically generated from the API and hence allow following the inheritance structure or MRO of the underlying classes in the software backend make it possible to extract knowledge graphs that follow an ontology directly constructed from the API respectively its software architecture.

All resources generated in this work, including the training and validation datasets, the fully trained neural networks, and the implementation of the node editor and the scripts for generating, visualizing, and analyzing the automatically generated knowledge graphs are made publicly available.

While the concept of creating the action graphs, node graphs, and knowledge graphs is very general, the final implementation and hence also the knowledge graphs and ontologies they follow are domain specific and hardware and software dependent. A common ontology for the different subdomains, such as nanomaterial synthesis, could help in the future to make knowledge graphs and even entire synthesis workflows platform independent and readily interoperable between different MAPs or SDLs with different software backends. Each SDL or MAP platform could map their hardware-specific process implementation to such a common description (controlled and shared vocabulary) or ontology, and every other platform could

map from the same description or ontology to their available hardware implementation and the interfaces exposed by their API. That way, every lab has to define these specific mapping steps only once and could then exchange synthesis protocols *via* this "common language". Another extension that seems natural in the context of node graphs and the generated knowledge graphs is the storage of the underlying data in graph databases, a step that is also planned to be further investigated in the future. Moreover, by using extracted knowledge graphs from a multitude of reactions, or even taking advantage of the quick inference times of the surrogate LLMs for building an extensive knowledge graph database from previously published procedures, it can be envisioned that these knowledge graphs can be used as input for AI models such as graph neural networks[26,27] or directed acyclic graph transformers[28] for learning new, causal relationships for inference.

Data availability

Data for this article, including the annotated datasets, the fully trained LLMs, the python code of the node editor, and a jupyter notebook for visualizing and querying the knowledge graphs are available on Github at **https://github.com/BAMresearch/MAPz_at_BAM/tree/main/Minerva-Workflow-Generator**, on huggingface at **https://huggingface.co/bruehle**, and on Zenodo (DOI: **https://doi.org/10.5281/zenodo.15228014**) at **https://zenodo.org/records/15228014**. Further data can also be found in the ESI.†

Conflicts of interest

There are no conflicts to declare.

Acknowledgements

We thank the HPC Center and IT Services at BAM for providing the computational resources used in this work. The Materials Acceleration Platform Center at BAM (MAPz@BAM) is gratefully acknowledged.

Notes and references

1 J. Wagner, C. G. Berger, X. Du, T. Stubhan, J. A. Hauch and C. J. Brabec, The evolution of Materials Acceleration Platforms: toward the laboratory of the future with AMANDA, *J. Mater. Sci.*, 2021, **56**, 16422–16446.

2 M. Abolhasani and E. Kumacheva, The rise of self-driving labs in chemical and materials sciences, *Nat. Synth.*, 2023, **2**, 483–492.

3 S. P. Stier, C. Kreisbeck, H. Ihssen, M. A. Popp, J. Hauch, K. Malek, M. Reynaud, T. p. m. Goumans, J. Carlsson, I. Todorov, L. Gold, A. Räder, W. Wenzel, S. T. Bandesha, P. Jacques, F. Garcia-Moreno, O. Arcelus, P. Friederich, S. Clark, M. Maglione, A. Laukkanen, I. E. Castelli, J. Carrasco, M. C. Cabanas, H. S. Stein, O. Ozcan, D. Elbert, K. Reuter, C. Scheurer, M. Demura, S. S. Han, T. Vegge, S. Nakamae, M. Fabrizio and M. Kozdras, Materials Acceleration Platforms (MAPs): Accelerating Materials Research and Development to Meet Urgent Societal Challenges, *Adv. Mater.*, 2024, **36**, 2407791.

4 L. Hawizy, D. M. Jessop, N. Adams and P. Murray-Rust, ChemicalTagger: A tool for semantic text-mining in chemistry, *J. Cheminf.*, 2011, **3**, 17.

5 D. M. Jessop, S. E. Adams, E. L. Willighagen, L. Hawizy and P. Murray-Rust, OSCAR4: a flexible architecture for chemical text-mining, *J. Cheminf.*, 2011, **3**, 41.

6 T. Rocktäschel, M. Weidlich and U. Leser, ChemSpot: a hybrid system for chemical named entity recognition, *Bioinformatics*, 2012, **28**, 1633–1640.

7 D. M. Lowe and R. A. Sayle, LeadMine: a grammar and dictionary driven approach to entity recognition, *J. Cheminf.*, 2015, **7**, S5.

8 R. Leaman, C.-H. Wei and Z. Lu, tmChem: a high performance approach for chemical named entity recognition and normalization, *J. Cheminf.*, 2015, **7**, S3.

9 M. Krallinger, O. Rabal, A. Lourenço, J. Oyarzabal and A. Valencia, Information Retrieval and Text Mining Technologies for Chemistry, *Chem. Rev.*, 2017, **117**, 7673–7761.

10 S. Mysore, E. Kim, E. Strubell, A. Liu, H.-S. Chang, S. Kompella, K. Huang, A. McCallum and E. Olivetti, Automatically Extracting Action Graphs from Materials Science Synthesis Procedures, *arXiv*, 2017, preprint, arXiv:1711.06872, DOI: **10.1021/acs.chemrev.6b00851**, **http://arxiv.org/abs/1711.06872**.

11 I. Korvigo, M. Holmatov, A. Zaikovskii and M. Skoblov, Putting hands to rest: efficient deep CNN-RNN architecture for chemical named entity recognition with no hand-crafted rules, *J. Cheminf.*, 2018, **10**, 28.

12 L. Weston, V. Tshitoyan, J. Dagdelen, O. Kononova, A. Trewartha, K. A. Persson, G. Ceder and A. Jain, Named Entity Recognition and Normalization Applied to Large-Scale Information Extraction from the Materials Science Literature, *J. Chem. Inf. Model.*, 2019, **59**, 3692–3702.

13 S. H. M. Mehr, M. Craven, A. I. Leonov, G. Keenan and L. Cronin, A universal system for digitization and automatic execution of the chemical synthesis literature, *Science*, 2020, **370**, 101–108.

14 A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, Attention is All you Need, *arXiv*, 2020, preprint, arXiv:1706.03762, DOI: **10.48550/arXiv.1706.03762**.

15 T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, in *Advances in Neural Information Processing Systems*, ed. H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin, Curran Associates, Inc., 2020, vol. 33, pp. 1877–1901.

16 A. C. Vaucher, F. Zipoli, J. Geluykens, V. H. Nair, P. Schwaller and T. Laino, Automated extraction of chemical synthesis

actions from experimental procedures, *Nat. Commun.*, 2020, **11**, 3601.
17 IBM RXN for Chemistry, **https://rxn.res.ibm.com/rxn/robo-rxn/welcome**, accessed 11 November 2024.
18 N. Yoshikawa, M. Skreta, K. Darvish, S. Arellano-Rubach, Z. Ji, L. Bjørn Kristensen, A. Z. Li, Y. Zhao, H. Xu, A. Kuramshin, A. Aspuru-Guzik, F. Shkurti and A. Garg, Large language models for chemistry robotics, *Auton. Robots*, 2023, **47**, 1057–1086.
19 Introducing Llama 3.1, **https://ai.meta.com/blog/meta-llama-3-1/**, accessed 9 August 2024.
20 D. Lowe, *Chemical reactions from US patents (1976–Sep 2016)*, 2017, figshare, Dataset, DOI: **10.6084/m9.figshare.5104873.v1**.
21 M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang and A. Ahmed, Big Bird: Transformers for Longer Sequences, *arXiv*, 2021, preprint, arXiv:2007.14062, DOI: **10.48550/arXiv.2007.14062**.
22 I. Beltagy, M. E. Peters and A. Cohan, Longformer: The Long-Document Transformer, *arXiv*, 2020, preprint, arXiv:2004.05150, DOI: **10.48550/arXiv.2007.14062**, **http://arxiv.org/abs/2004.05150**.
23 M. Zaki, C. Prinz and B. Ruehle, A Self-Driving Lab for Nano- and Advanced Materials Synthesis, *ACS Nano*, 2025, **19**, 9029–9041.
24 google/bigbird-pegasus-large-bigpatent · Hugging Face, **https://huggingface.co/google/bigbird-pegasus-large-bigpatent**, accessed 9 August 2024.
25 allenai/led-base-16384 · Hugging Face, **https://huggingface.co/allenai/led-base-16384**, accessed 9 August 2024.
26 Z. Ye, Y. J. Kumar, G. O. Sing, F. Song and J. Wang, A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs, *IEEE Access*, 2022, **10**, 75729–75741.
27 H. Jin, K. Raghavan, G. Papadimitriou, C. Wang, A. Mandal, M. Kiran, E. Deelman and P. Balaprakash, Graph neural networks for detecting anomalies in scientific workflows, *Int. J. High Perform. Comput. Appl.*, 2023, **37**, 394–411.
28 J. Yu, M. Gao, Y. Li, Z. Zhang, W. H. Ip and K. L. Yung, Workflow performance prediction based on graph structure aware deep attention neural network, *J. Ind. Inf. Integr.*, 2022, **27**, 100337.