

Modular Design Patterns for Generative Neuro-Symbolic Systems

Maaïke H. T. de Boer^{1,*}, Quirine S. Smit^{1,*}, Michael van Bekkum¹,
André Meyer-Vitali² and Thomas Schmid^{3,4,5}

¹TNO, dep. Data Science, The Hague, The Netherlands

²Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), Saarbrücken, Germany

³Martin Luther University Halle-Wittenberg, Halle (Saale), Germany

⁴Lancaster University in Leipzig, Leipzig, Germany

⁵Leipzig University, Leipzig, Germany

Abstract

Developing systems that are able to generate novel outputs is one of the dominating trends in current Artificial Intelligence (AI) research. Both capabilities and availability of such generative systems, in particular of so-called Large Language Models (LLMs), have been exploding in recent years. While Neuro-Symbolic generative models offer advantages over purely statistical generative models, it is currently difficult to compare the different ways in which the training, fine-tuning and usage of the growing variety of such approaches is carried out. In this work, we use the modular design patterns and Boxology language of van Bekkum et al for this purpose and extend those to enable the representation of generative models, specifically LLMs. These patterns provide a general language to describe, compare and understand the different architectures and methods used. Our main aim is to support better understanding of generative models as well as to support engineering of LLM-based systems. In order to demonstrate the usefulness of this approach, we explore generative Neuro-Symbolic architectures and approaches as use cases for these generative design patterns.

Keywords

design patterns, neuro-symbolic AI, generative models, Large Language Models

1. Introduction


Recently, Artificial Intelligence (AI) has taken a leap in the form of generative models. Prominently, multimodal statistical models, such as DALL-E [1] and Stable Diffusion [2] have changed the world of image generation, and with the release of OpenAI's ChatGPT system¹, the world of text generation has changed forever. Targeting text generation tasks in particular, both the development and the number of Large Language Models (LLMs) has increased enormously. Currently, many different generative models are popping up, both open-source and proprietary [3]. Moreover, due to open challenges of LLMs, such as hallucination [4], explainability [5] and trustworthiness, novel Neuro-Symbolic generative approaches have emerged [6, 7].

GeNeSy'24: First International Workshop on Generative Neuro-Symbolic AI, May 2024, Hersonissos, Crete, Greece

*Corresponding author; both authors contributed equally.

✉ maaïke.deboer@tno.nl (M. H. T. d. Boer); quirine.smit@tno.nl (Q. S. Smit); andre.meyer-vitali@dfki.de (A. Meyer-Vitali); thomas.schmid@medizin.uni-halle.de (T. Schmid)

ORCID 0000-0002-2775-8351 (M. H. T. d. Boer); 0000-0002-5242-1443 (A. Meyer-Vitali)

 © 2024 Copyright (c) 2024 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://chat.openai.com/chat>

Not only several LLMs, but also a large number of so-called foundation models dealing with various input and output modalities have entered the scene in recent years. Due to the quantity and diversity of emerging generative techniques, it becomes more and more challenging to keep track of the ever-growing variety of models with different architectures and capabilities. One of the solutions to tackle this issue is to create a high-level conceptual framework to discuss, compare, configure and combine different models using a Boxology. The Boxology started in the field of Neuro-Symbolic systems, by Van Harmelen and Ten Teije [8] in 2019. This work is extended in 2021 by van Bekkum et al. [9] by providing a taxonomically organised vocabulary to describe both processes and data structures used in hybrid systems.

Here, we propose to use and extend the Boxology to gain insights in a variety of generative models, specifically on LLMs. To this end, we test validity and usefulness of the Boxology in this field on example architectures and applications, such as ChatGPT, KnowGL, GENOME and Logic-LM. Our modular approach supports new architectures and engineering approaches to systems based on generative AI models. Our pattern extensions promote transparency and trustworthiness in system design, by providing interpretable, high-level component descriptions of generative AI models.

The rest of the paper is organized as follows. In the next section, we give a more detailed overview of the Boxology. In the third section, we propose to extend the Boxology by three novel patterns in order to be able to handle generative models. In section 4, we dive into specific applications and tasks in which generative models, specifically in Neuro-Symbolic systems, are used. We conclude with summarizing our key findings and outlining future work.

2. Related Work on the Boxology

We will base our paper on the paper by van Bekkum et al. [9], in which the authors provide a taxonomically organised vocabulary to describe both processes and data structures used in hybrid systems. The highest level of this taxonomy contains instances, models, processes and actors, which may be described as follows.

Instances: The two main classes of instances are data and symbols. *Symbols* are defined as to have a designation to an object, class or a relation in the world, which can be either atomic or complex, and when a new symbol is created from another symbol and a system of operations, it should have a designation. Examples of symbols are labels (short descriptions), relations (connections between data items, such as triples) and traces (records of data and events). *Data* is defined as not symbolic. Examples are numbers, texts, tensors or streams.

Models: Models are descriptions of entities and their relationships, which can be statistical or semantic. *Statistical* models represent dependencies between statistical variables, such as LLMs or Bayesian Networks. *Semantic* models specify concepts, attributes and relationships to represent the implicit meaning of symbols, such as ontologies, taxonomies, knowledge graphs or rule bases.

Processes: Processes are operations instances and models. Three types of processes are defined: generation, transformation and inference. *Generation* can be done using, for example, the

training of a model or by knowledge engineering. *Transformation* is the transformation of data, for example from knowledge graph to vector space. *Inference* can be inductive or deductive, in which induction generalises instances and deduction reaches conclusions on specific instances, such as with classification.

Actors: Actors can be humans, (software) agents or robots (physically embedded agents). Meyer-Vitali et al. [10] extended the original paper with a definition of teams of actors in the Boxology.

Besides the vocabulary, the visual language is defined in van Bekkum et al. [9], as an extension on Van Harmelen and Ten Teije [8]. The visual language consists of rectangular boxes (instances), hexagonal boxes (models), ovals (processes) and triangles (actors) and unspecified arrows between them. Within the boxes the concept will be noted by each level in the vocabulary using colon-separation from most generic to most-specific, for example a neural network will be model:stat:NN.

van Bekkum et al. [9] present elementary patterns, which can then be combined into more complex patterns. Patterns 1a and 2a from Figure 1, for example, can be combined into a pattern which is named 3a in the paper (depicted in Figure 2). Whereas 1a describes the pattern of training a model based on data (data generates a model), 2a describes the usage of the model in deducing a symbol (data and model deduce a symbol), such as a prediction. The combination in 3a describes a basic structure for a (statistical) Machine Learning (ML) model depicting the training (creating the model) and testing or application phase (applying the model on new data).

In the past years, the Boxology has been used and extended in different ways. Three of the most influential papers are the formalisation of the notions from the Boxology and implementation in the heterogeneous tool set (Hets) [11], the extension of the Boxology for (teams of) actors [10] and the systematic study of nearly 500 papers published in the past decade in the area of Semantic Web Machine Learning [12].

3. Design Patterns for Generative Models

While Generative AI originates in the realm of data-driven AI, it has demonstrated capabilities that exceed classical machine learning tasks like classification and regression by far. In particular, such generative systems specialise in the generation of content, such as images [1, 2], videos [13], or text [14, 15, 16]. In the original, purely statistical setting, these capabilities are acquired during a so-called (pre-)training phase [17] where a representation of a large data body is learned and in a second phase used to process input to output that has not explicitly been specified but follows the characteristics of the data body (application phase).

However, specific arrangements for both (pre-)training and representation usage in downstream tasks vary for different approaches and systems [18]. In order to allow for a coherent description of the generative paradigm, we propose to extend the elementary patterns of van Bekkum et al. [9] that describe the generic pattern for instances, models, processes and actors (Figure 1 1a-1d and 2a-d). Please note that while patterns 1e and 1f are required for certain aspects of the generative paradigm, their usage is not limited to this. Data generation and labelling by humans may also be employed work with any statistical approach.

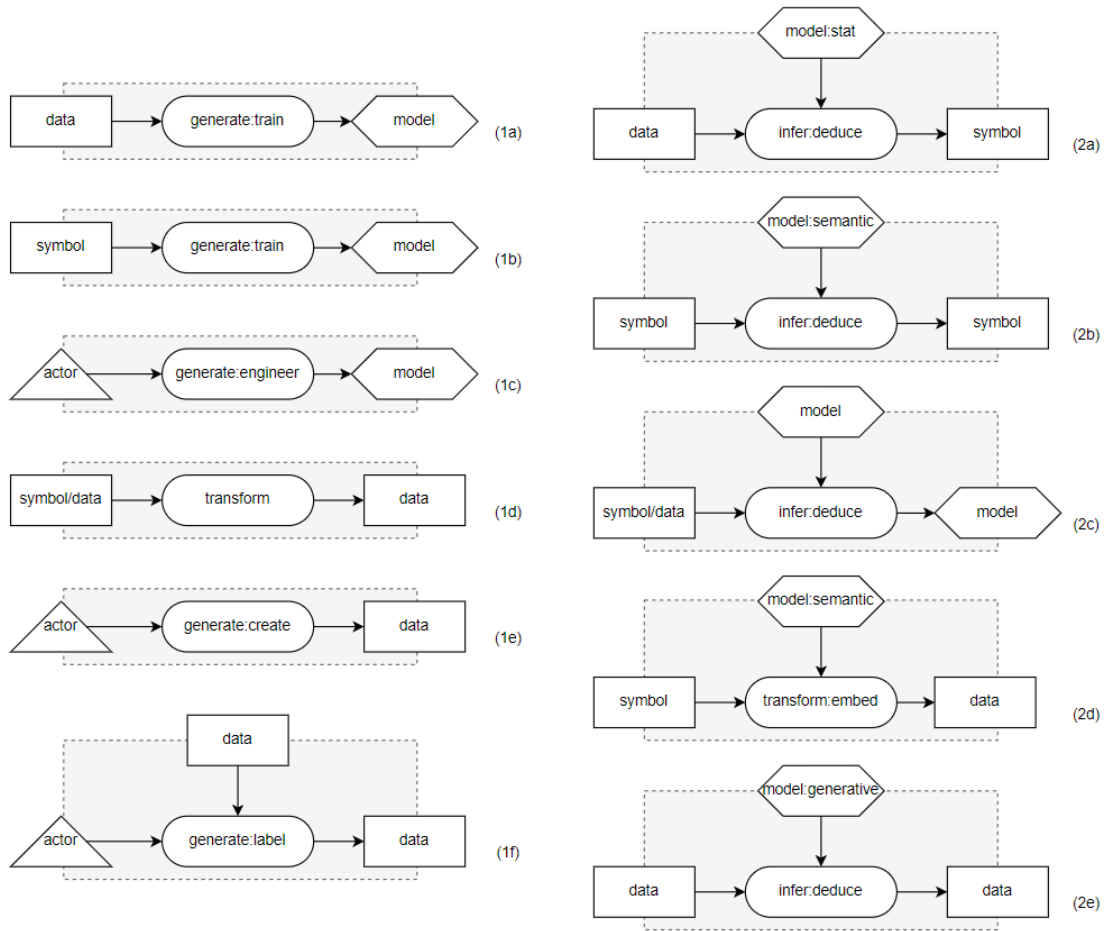


Figure 1: All elementary design patterns, including proposed additions 1e, 1f and 2e

In particular, when describing classical machine learning systems, mostly pattern 2a is used, where the output is a symbol, such as a classification or a label. However, the key concept in generative models is that the output is not a symbol, but data; this can be an image, video or text, depending on the model. Additionally, actors play an important role in Generative AI, by creating prompts or label data. To this end, we here propose three new elementary patterns: pattern 1e, in which an actor can generate data, pattern 1f, in which an actor labels data, and 2e, in which a model can deduce data from data. In the remainder of this section we mainly focus on Large Language Models (LLMs). Please note, however, that the patterns proposed in this section are transferable to other data types, for example to vision transformers, which follow a similar architecture paradigm as transformers but operate on image data .

3.1. Transformer Models

The key technology behind basically all current LLMs is the so-called transformer architecture. The original transformer paper by Vaswani et al. [19] proposed to use two interacting models,

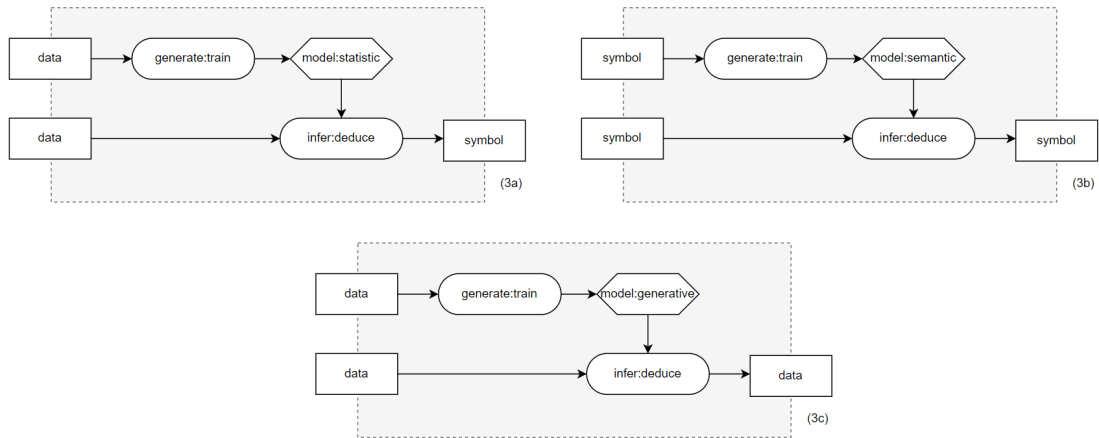


Figure 2: Compositional design patterns, including proposed addition 3c made by combining elementary pattern 1a and 2e.

an encoder and a decoder. In the transformer family, some models, however, only use the encoder or the decoder part [18]. Figure 3A shows the architecture of a transformer model as a design pattern. Transformers are made up of two parts, an encoder and a decoder. These are usually trained end-to-end (such as flan-T5 [20]), but can also be used separately as encoder-only (Figure 3B) or decoder-only (Figure 3C) models. In the following sections, we focus on an encoder-only and a decoder-only family. Other sections focus on instructions and prompting of different models and the interaction with actors.

3.1.1. Encoder only: BERT (base)

Some systems are encoder-only. These systems are specialised in contextual encoding, often named a base model. They can ‘understand’ and encode input sentences. An encoder model is trained using data, pattern 1a. It is often connected to other systems, such as a classification system, pattern 3a (see Figure 3B), to be useful for tasks other than the encoding input sentences. An example of this is BERT [21]. Encoders are transformer models, but not generative models.

3.1.2. Decoder only: GPT

Other transformer based systems have decoder-only architectures. This approach is complementary to the encoder-only paradigm, but structurally different [18]: an encoder processes the input data (in these cases text) and transforms it into a different, machine interpretable, representation, often a vector representation. A decoder-only system, on the other hand, decodes the input data directly, without being transformed into a higher, more abstract representation, to the desired representation (text or images). Examples of this are generative models from the GPT family [14].

In the Boxology, both encoders and decoders have a similar representation. For generative models from the GPT family, we suggest pattern 3c, (see Figure 2), which is a combination of 1a and 2e, as presented in Figure 1: data is used to train a decoder model, which does not use an

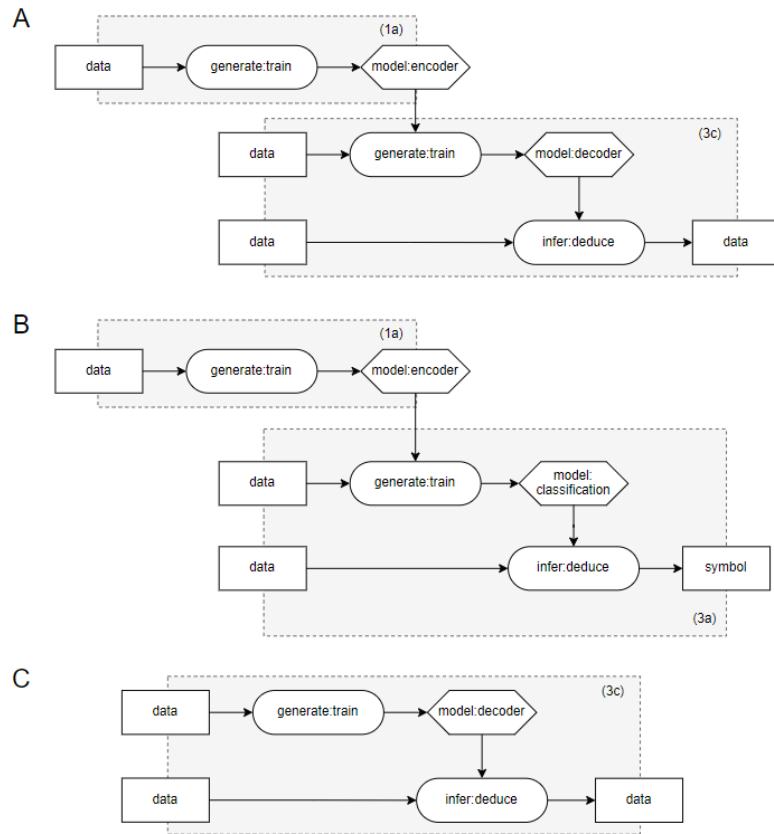


Figure 3: Three uses of transformers. A shows the traditional encoder-decoder architecture. B shows an encoder-only model applied to a classification task. C shows a decoder-only architecture.

encoder as input as well, such as with other transformers. This decoder model can be used to deduce output data from input data directly.

Decoder-only architectures may be further divided into causal decoder architectures and prefix decoder architectures. Causal decoder architectures, such as GPT [22, 14] and BLOOMZ [23], use only unidirectional attention to the input sequence by using a specific mask. Prefix decoder architectures, such as PaLM [24], uses the bidirectional attention for tokens in the prefix while maintaining unidirectional attention for generating subsequent tokens. Both architectures follow the elementary pattern 2e.

3.1.3. Prompts and Instructions

One of the main differences between current LLMs and earlier BERT or other transformer models is that the model is fine-tuned on instructions [18]. Multi-task fine-tuning or instruction tuning, is currently often done using a collection of datasets phrased as instructions, to improve model performance and generalisation to unseen tasks [20]. The original model is often referred to as foundation model [25], whereas the fine-tuned model is an adjusted model. In the Boxology,

we define this adjusted model as another model as we did with the encoder and decoder model in Figure 3, but then stacking two decoder models. This instruction tuning also follows pattern 1a, but this data is different as it also contains instructions.

Next to instruction learning LLMs can also be tweaked by in-context learning. Here examples are used as part of the prompt to give context for the answers to the instructions. In this case the model weights are not changed. This optimizes the performance of models on different tasks [26], but does not need as much training data as training a model from scratch. These prompts can include a few (training) examples of the input and output (few-shot) or no examples (zero-shot). These few-shot examples do not train the foundational or instruction model, and therefore we model them as input data that is used to deduce data (text), which is pattern 2e. Assistants or GPTs could, however, be seen as a new model, especially if they perform other tasks, such as Retrieval Augmented Generation (RAG).

3.2. Actor Interaction

Actors play a large role in the current generative models. In the original paper by van Bekkum et al. [9], patterns using actors are underspecified. On the one hand, actors often create data, not only in the interaction with an agent that uses generative models, but also in common Machine Learning approaches. Many of the created textual datasets are written, pre-processed and labelled by actors. A first proposed pattern is pattern 1e, in which an actor creates data. The second proposed pattern is pattern 1f, in which an actor generates a label, or annotates data. Both patterns are depicted in Figure 1.

Generative models are often not used only once. With the current chat functions, actors are interacting with the model multiple times. The main difference with other Machine Learning models, where also more data is inputted and symbols are outputted, the data inputted is often not dependent on the output of the previous data point. However, with conversational generative models, prompts can be related to the previous response. Currently, recurrent or iterative behaviour is not yet part of the pattern concepts.

4. Design Patterns for Generative Neuro-Symbolic AI

In this section, we describe and explore several papers that use generative models in a Neuro-Symbolic system. The selected papers are chosen, as they represent a diverse set of possibilities to use a generative model, at the start of the system, in the middle and at the end, but also to act as a fluent language interface or a formal language interface. We also included ChatGPT, which is the most famous generative AI system, and although mainly data driven, includes a symbolic component in the reward modelling part of the training phase.

4.1. (Training of) ChatGPT

ChatGPT is an application of the foundational model GPT3 [14], and later GPT4 [27]. It is trained further to be of aid in the setting of an assistant. The architecture of the training phases is represented in Figure 4. The foundational model GPT3 is used as a basis for further training

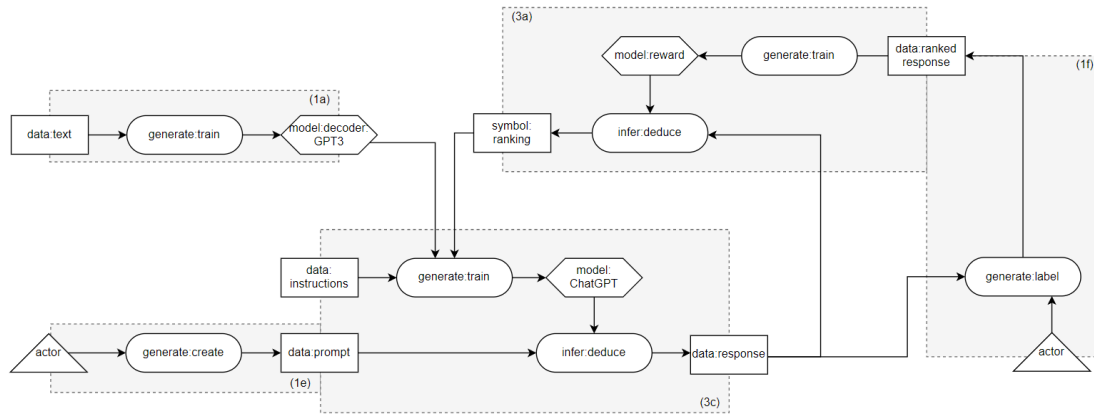


Figure 4: Training phase of ChatGPT

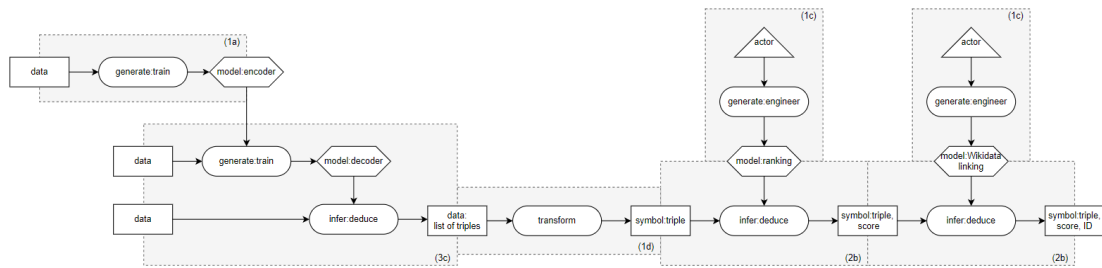


Figure 5: Boxology representation of KnowGL [28]

(1a). Instructions and answers are used to train what will become ChatGPT. Then, based on new prompts the model generates a response (3c).

To further train ChatGPT to give the desired responses the reward model is added. The reward model is a separate model, which can judge if a response is a good one, given the instructions. The reward model is trained by people annotating the multiple answers to instructions. To train the reward model, the model trained on instructions is asked to output multiple answers. These answers are then ranked by annotators to generate a training set for the reward model (1f). The reward model is trained to compare answers of ChatGPT and return their score (3a). This is then used in a loop with the ChatGPT to improve the instruction answering process. As one can view, we have adapted Boxology patterns to be able to accept multiple inputs.

When applying ChatGPT in a pipeline, it suffices to show only pattern 3c, the block containing ChatGPT and 1e to show the user writing the prompt.

4.2. KnowGL

Figure 5 shows KnowGL Parser [28], a NeSy system combining a generative module and symbolic methods. The KnowGL Parser can be used to automatically extract knowledge graphs from collections of documents. It is based on BART-large, which has an encoder-decoder architecture.

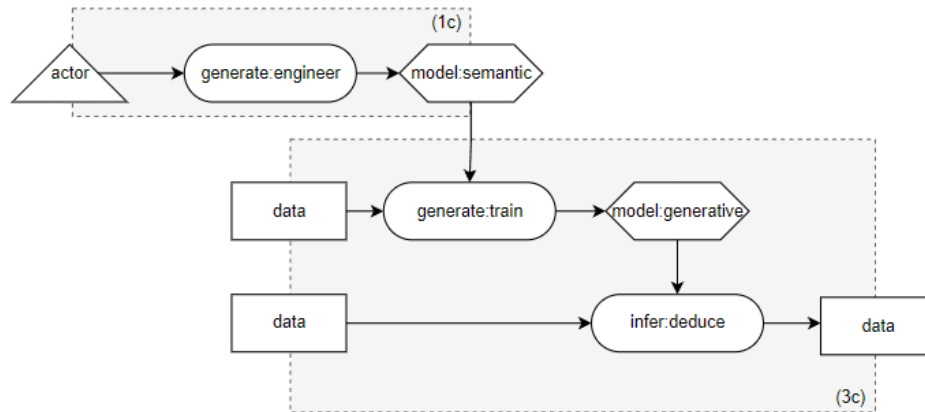


Figure 6: Boxology representation of KnowBERT [29]

The encoder receives a sentence (1a) and the decoder generates a list of ‘subject, relation, object’ (3c). These are then parsed (transformed) in preparation of the next step, fact ranking (1d). Here a ranked list is created of distinct facts and their scores (2b). In the final step the generated facts are linked to Wikidata. This is done using a mapping of labels to Wikidata IDs (2b). In the case that the generative model has created a new entity, type or relation label that are not in Wikidata it returns ‘null’.

4.3. KnowBERT

While knowledge is mostly injected to statistical generative models either during the input or during the output stage, also approaches to inject knowledge inside the model have been proposed. A prominent example is KnowBERT, a modified variant of the transformer architecture BERT [29]. Although not a generative model, it stands out for its fusion of contextual and graph representations, attention-enhanced entity spanned knowledge infusion, and flexibility in injecting multiple Knowledge Graphs at various model levels. By integrating so-called Knowledge Attention and Recontextualization (KAR) layers [30], graph entity embeddings are utilized that are processed through an attention mechanism to enhance entity span embeddings. This happens in later layers of the model to stabilize training but may potentially also used to inject knowledge at earlier stages [6]. The Boxology pattern for KnowBERT is depicted in Figure 6.

4.4. Mathematical Conjecturing and LLMs

The system proposed by Johansson and Smallbone [31] assigns the generative task of discovery of mathematical conjectures to a LLM (3c), while the results can be checked afterwards using a symbolic theorem prover or counter-example finder (2b). The system is prompted with a formal theory (e.g. a sort function), and has the LLM generate lemmas from the theory. These generated lemmas are transformed from data to symbol and can then be used by the semantic model(s). The pattern is depicted in Figure 7. The approach taken in Yang et al. [32] is also captured by this pattern. The system proposed uses a LLM component to produce Prolog code

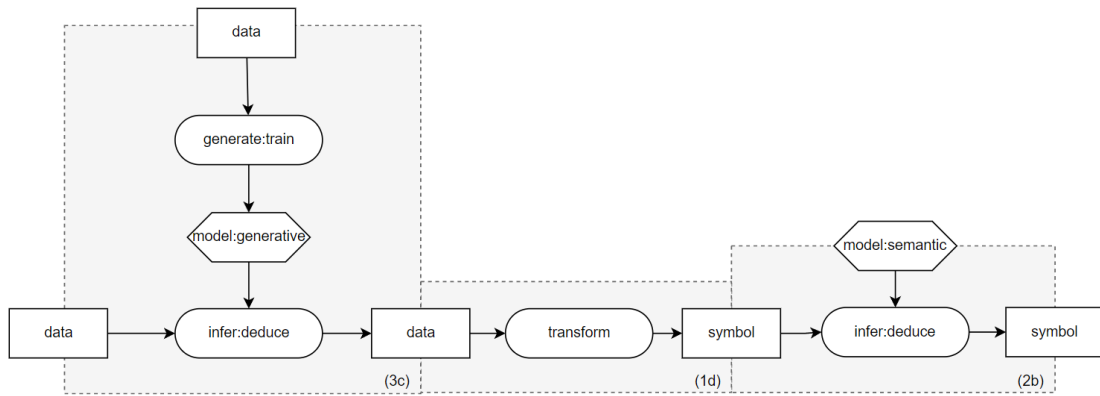


Figure 7: Boxology representation of using LLMs for discovery of mathematical conjectures [31]

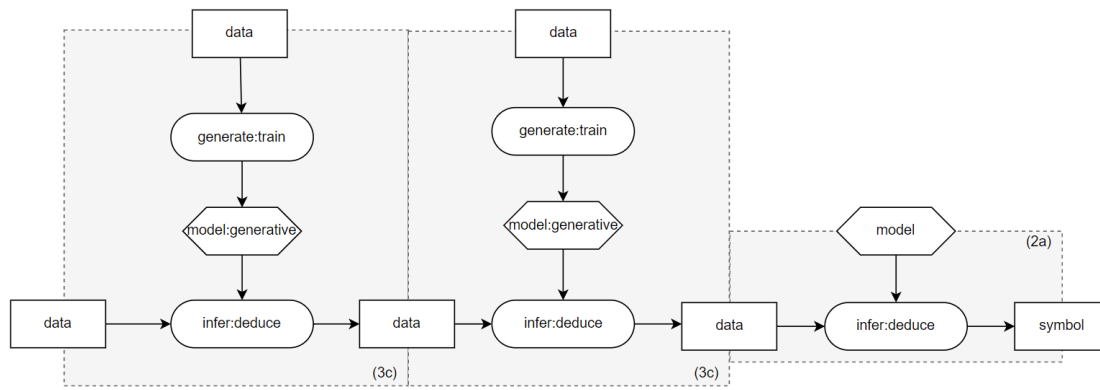


Figure 8: Boxology representation of GENOME [33]

(3c) and a symbolic inference engine to produce answers and reasoning traces by executing the aforementioned code (1d, 2b).

4.5. GENOME

Generative Neuro-Symbolic Visual Reasoning by Growing and Reusing Modules (GENOME) [33] focuses on the task of generative software module learning, based on a LLM generating signatures (input/output) and reasoning steps, then have a LLM create the software module based on those and evaluate the module on test cases.

The system consists of three stages: module initialization, module generation, and module execution. The design pattern is depicted in Figure 8. First a LLM assesses a visual-language question and outputs new module signatures and operation steps as a response to the query (3c), if current modules cannot provide an adequate response. In the next step, the LLM creates a module (software code) based on the signature/test case (3c). Finally the module is executed by passing it a visual query (2a).

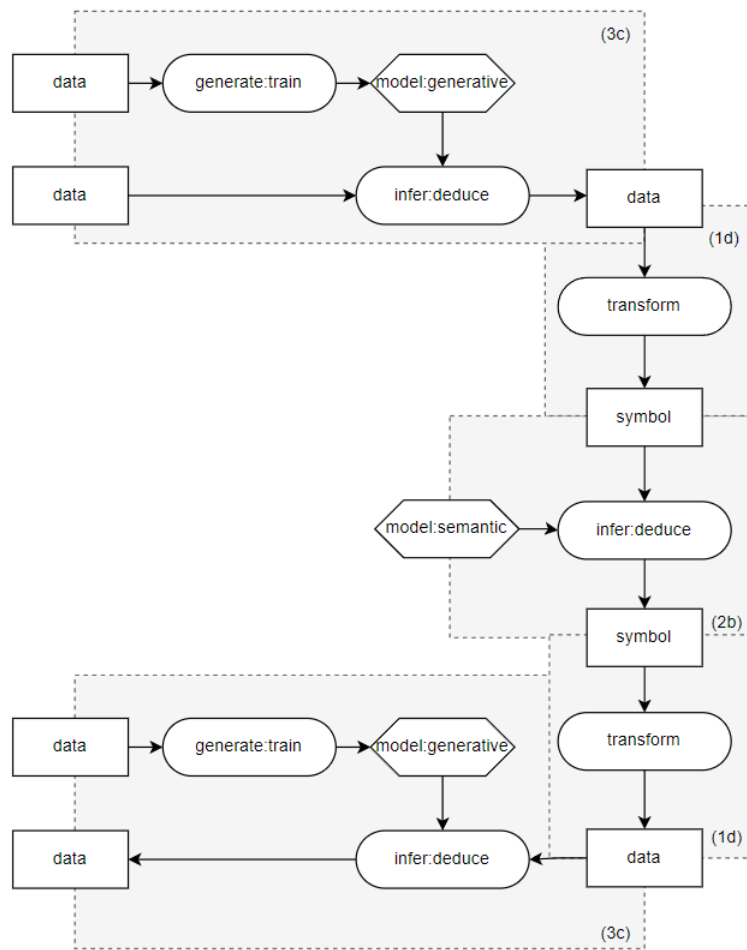


Figure 9: Boxology representation of Logic-LM [34]

4.6. Logic-LM

Logic-LM [34] integrates LLMs with symbolic solvers to improve logical problem-solving. The pattern is depicted in Figure 9: the system utilizes LLMs to translate a problem stated in natural language problem into a symbolic formulation (3c). In the next step, a symbolic reasoner performs logical inference on the formulated problem (1d, 2b, 1d). Finally, an LLM interprets the results and outputs natural language (3c). The LLM thus functions as a fluent language interface (both on input and output) to a symbolic reasoner component.

5. Conclusion and Future Work

Generative AI is currently a major technology with many applications and combining data-driven approaches with knowledge-based techniques is a promising development to this end. In this paper, we propose new design patterns for modular generative Neuro-Symbolic systems to

be included into the design pattern approach for Neuro-Symbolic systems as proposed by van Bekkum et al. [9]. We show how the composition of elementary patterns can be used to describe generative models, and we explore several specific generative models, such as ChatGPT, as well as several generative NeSy papers, such as KnowGL, GENOME and Logic-LM.

We acknowledge that this is only the first step in a more elaborate exploration on generative design patterns and the description of generative Neuro-Symbolic architectures. In future work, we would like to validate our proposals for extending the Boxology, by applying them to more examples from additional papers. In addition, we expect to further extend and deepen the Boxology itself. In this paper, it became clear that the temporal or iterative aspect is not yet visualised well, as well as the naming and formalisation of the Boxology, including the do's and don'ts: which pattern combinations are allowed and which are not? The importance of modelling datasets for generative AI may be taken into account in future specifications of particular subtypes of Instances and Models in the taxonomy. Additionally, the use of graphical tools for software development is well-known from the Unified Modelling Language (UML) and visual programming tools, such as LabView or Scratch. We are mostly concerned with graphical representations of design patterns for system design and documentation, but the promise of templates, low-code or no-code development is appealing for the future.

Acknowledgements

We would like to thank the TNO project GRAIL for their financial support, as well as Frank van Harmelen and Annette ten Teije for their feedback. We would also like to thank Daan Di Scala for his contribution to the KnowGL pattern.

References

- [1] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo, et al., Improving image generation with better captions, *Computer Science* 2 (2023) 8.
- [2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution image synthesis with latent diffusion models, 2021. [arXiv:2112.10752](https://arxiv.org/abs/2112.10752).
- [3] H. Chen, F. Jiao, X. Li, C. Qin, M. Ravaut, R. Zhao, C. Xiong, S. Joty, Chatgpt's one-year anniversary: Are open-source large language models catching up?, [arXiv:2311.16989](https://arxiv.org/abs/2311.16989) (2023).
- [4] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, *ACM Comput. Surv.* 55 (2023).
- [5] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, M. Du, Explainability for large language models: A survey, *ACM Transactions on Intelligent Systems and Technology* 15 (2024) 1–38.
- [6] P. Colon-Hernandez, C. Havasi, J. Alonso, M. Huggins, C. Breazeal, Combining pre-trained language models and structured knowledge, [arXiv preprint arXiv:2101.12294](https://arxiv.org/abs/2101.12294) (2021).
- [7] X. Wei, S. Wang, D. Zhang, P. Bhatia, A. Arnold, Knowledge enhanced pretrained language models: A comprehensive survey, [arXiv:2110.08455](https://arxiv.org/abs/2110.08455) (2021).

- [8] F. Van Harmelen, A. Ten Teije, A boxology of design patterns for hybrid learning and reasoning systems, *Journal of Web Engineering* 18 (2019) 97–123.
- [9] M. van Bekkum, M. de Boer, F. van Harmelen, A. Meyer-Vitali, A. t. Teije, Modular design patterns for hybrid learning and reasoning systems: a taxonomy, patterns and use cases, *Applied Intelligence* 51 (2021) 6528–6546.
- [10] A. Meyer-Vitali, W. Mulder, M. H. T. de Boer, Modular design patterns for hybrid actors, 2021. [arXiv:2109.09331](https://arxiv.org/abs/2109.09331).
- [11] T. Mossakowski, Modular design patterns for neural-symbolic integration: refinement and combination, [arXiv:2206.04724](https://arxiv.org/abs/2206.04724) (2022).
- [12] A. Breit, L. Waltersdorfer, F. J. Ekaputra, M. Sabou, A. Ekelhart, A. Iana, H. Paulheim, J. Portisch, A. Revenko, A. t. Teije, et al., Combining machine learning and semantic web: A systematic mapping study, *ACM Computing Surveys* (2023).
- [13] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, et al., Sora: A review on background, technology, limitations, and opportunities of large vision models, [arXiv:2402.17177](https://arxiv.org/abs/2402.17177) (2024).
- [14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, *Advances in neural information processing systems* 33 (2020) 1877–1901.
- [15] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, 2023. [arXiv:2302.13971](https://arxiv.org/abs/2302.13971).
- [16] S. Pichai, D. Hassabis, Introducing gemini: our largest and most capable ai model, 2023.
- [17] D. Erhan, A. Courville, Y. Bengio, P. Vincent, Why does unsupervised pre-training help deep learning?, in: *Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings*, 2010, pp. 201–208.
- [18] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, D. Roth, Recent advances in natural language processing via large pre-trained language models: A survey, *ACM Computing Surveys* 56 (2023) 1–40.
- [19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, *Advances in neural information processing systems* 30 (2017).
- [20] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, et al., Scaling instruction-finetuned language models, [arXiv:2210.11416](https://arxiv.org/abs/2210.11416) (2022).
- [21] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. [arXiv:1810.04805](https://arxiv.org/abs/1810.04805).
- [22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, *OpenAI blog* 1 (2019) 9.
- [23] N. Muennighoff, T. Wang, L. Sutawika, A. Roberts, S. Biderman, T. L. Scao, M. S. Bari, S. Shen, Z.-X. Yong, H. Schoelkopf, et al., Crosslingual generalization through multitask finetuning, [arXiv:2211.01786](https://arxiv.org/abs/2211.01786) (2022).
- [24] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al., Palm: Scaling language modeling with pathways, *Journal of Machine Learning Research* 24 (2023) 1–113.
- [25] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein,

- J. Bohg, A. Bosselut, E. Brunskill, et al., On the opportunities and risks of foundation models, arXiv:2108.07258 (2021).
- [26] Y. Liu, H. He, T. Han, X. Zhang, M. Liu, J. Tian, Y. Zhang, J. Wang, X. Gao, T. Zhong, et al., Understanding llms: A comprehensive overview from training to inference, arXiv:2401.02038 (2024).
- [27] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, Y. Tang, A brief overview of chatgpt: The history, status quo and potential future development, 2023.
- [28] G. Rossiello, M. F. M. Chowdhury, N. Mihindukulasooriya, O. Cornec, A. M. Gliozzo, Knowgl: Knowledge generation and linking from text, in: AAAI, 2023, pp. 16476–16478.
- [29] M. E. Peters, M. Neumann, R. L. Logan IV, R. Schwartz, V. Joshi, S. Singh, N. A. Smith, Knowledge enhanced contextual word representations, arXiv:1909.04164 (2019).
- [30] I. Balažević, C. Allen, T. M. Hospedales, Tucker: Tensor factorization for knowledge graph completion, arXiv:1901.09590 (2019).
- [31] M. Johansson, N. Smallbone, Exploring mathematical conjecturing with large language models, Proceedings of NeSy (2023).
- [32] S. Yang, X. Li, L. Cui, L. Bing, W. Lam, Neuro-symbolic integration brings causal and reliable reasoning proofs, 2023. arXiv:2311.09802.
- [33] Z. Chen, R. Sun, W. Liu, Y. Hong, C. Gan, Genome: Generative neuro-symbolic visual reasoning by growing and reusing modules, 2023. arXiv:2311.04901.
- [34] L. Pan, A. Albalak, X. Wang, W. Y. Wang, Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning, arXiv:2305.12295 (2023).