Introducing VULCAN:
A Visualization Tool for Understanding our Models and Data by Example

Jonas Groschwitz
University of Amsterdam*
j.d.groschwitz@uva.nl

Abstract

Examples are a powerful tool that help us understand complex concepts and connections. In computational linguistics research, looking at example system output and example corpus entries can offer a wealth of insights that are not otherwise accessible. This paper describes the open-source software VULCAN, a visualization tool for strings, graphs, trees, alignments, attention and more. VULCAN's unique ability to visualize both linguistic structures and properties of neural models make it particularly relevant for neuro-symbolic models. Neuro-symbolic models, combining neural networks with often linguistically grounded structures, offer a promise of increased interpretability in an age of purely neural black-box end-to-end models. VULCAN aims to facilitate this interpretability in practice. VULCAN is designed to be both easy to use and powerful in its capabilities.

1 Introduction

Humans effortlessly abstract patterns and complex generalizations from even small sets of examples (Posner and Keele, 1968; Gick and Holyoak, 1983; Brown and Kane, 1988). This ability is particularly useful for researchers in the field of NLP: Looking at entries in a corpus helps us understand what kinds of structures and phenomena occur in the corpus, and looking at example model outputs helps us understand the abilities and limitations of the model. This empowers us to form a more precise notion of how much we can (or cannot) trust the model to be correct, and it often provides us with ideas for how the model can be improved.

Here we introduce VULCAN: Visualizations for Understanding Language Corpora And model predictioNs. VULCAN is a flexible, user-friendly tool for visualizing linguistic structures and NLP model predicions. For example, Fig. 1 shows VULCAN visualizing an entry from the Little Prince Abstract Meaning Representation (AMR; Banarescu et al., 2013) corpus.[1] It shows both the gold AMR annotation (left) and an AMR predicted by a semantic parser (right).

Beyond visualizing linguistics structures like strings, trees and graphs, VULCAN can also display information such as alternative model predictions, attention (including transformer's multi-head attention; Vaswani et al., 2017) and alignments.

Thus, VULCAN is particularly useful for interpreting *neuro-symbolic* models, which combine neural networks with explicit (often linguistically grounded) reasoning and structures. One selling point of neuro-symbolic models is their interpretability. VULCAN, with its unique ability to display both linguistic structures and properties of neural models, makes that interpretability accessible in practice.

VULCAN also includes an advanced search functionality to find examples relevant to a certain research question or problem statement. The software is designed to be easy to use, and to be applicable to a wide range of corpora and models. VULCAN is available open source.[2]

2 Related Work

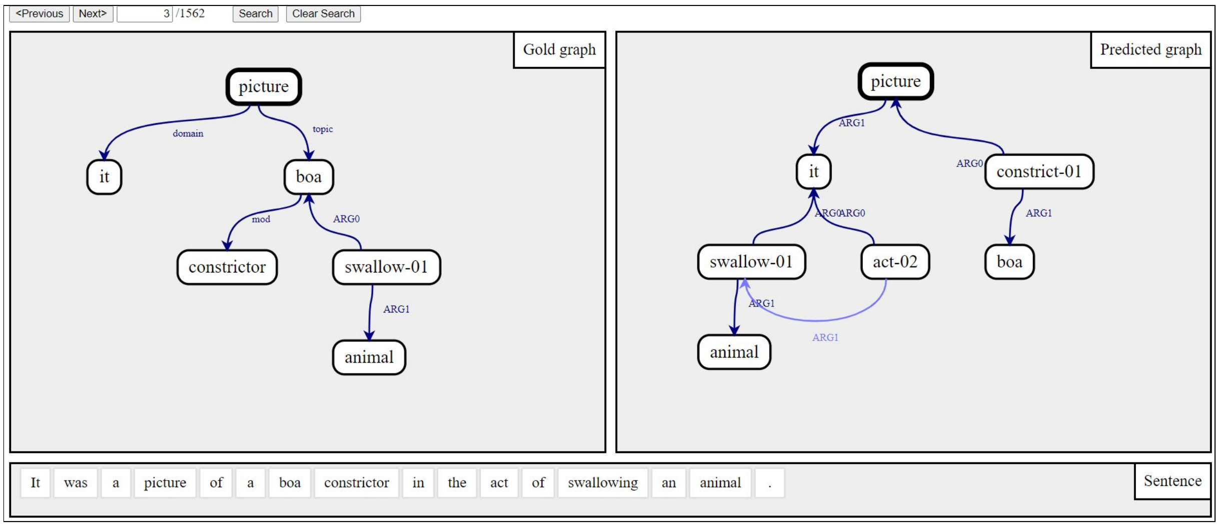
Many NLP visualization tools exist. However, they tend to be limited to a smaller range of linguistic objects they can display, are often tied to a specific framework, and don't always allow displaying model properties such as attention. We provide a selection of examples here:

- AllenNLP Interpret (Wallace et al., 2019) is a flexible tool to visualize neural models' predictions, including features such as visualizing attention. However, AllenNLP Interpret does not visualize more complex objects such

* Work done in part at the University of Edinburgh

[1] https://amr.isi.edu/download/amr-bank-struct-v3.0.txt

[2] https://github.com/jgroschwitz/vulcan



It was a picture of a boa constrictor in the act of swallowing an animal .

Figure 1: VULCAN visualizing a sentence, the corresponding gold Abstract Meaning Representation (AMR) semantic graph (left) and a predicted AMR (right).

as trees and graphs. Further, it is closely tied to the AllenNLP library (Gardner et al., 2018), development of which has been discontinued.

- Spacy's Displacy[3] is limited to POS tags and dependency trees. It does not show e.g. alternative model predictions.
- NeAt-vision[4] and VisuaLLM (Trebuňa and Dusek, 2023) visualize model information, such as alternative predictions including their likelihoods,but only support strings.
- MaltEval (Nilsson and Nivre, 2008) allows comparison between gold annotation and model prediction, but is limited to dependency trees.
- AMR-utils[5] visualizes AMR graphs only.
- Many corpora have their own online visualization. For example for the Universal Dependencies (de Marneffe et al., 2021) treebanks, multiple examples can be found.[6] VULCAN makes basic corpus browsing functionality available out of the box.

In sum, VULCAN fills a gap by visualizing both linguistic structures and (neural) model properties, a crucial combination for interpreting neuro-symbolic systems.

There are also more general visualization tools available. This includes LaTeX packages like TikZ,[7] and e.g. the graph plotting software Graphviz.[8] Their broad functionality, that usually goes far beyond NLP, can be a hurdle for learning how to use them effectively and can make them cumbersome to use even for experts. By contrast, VULCAN is designed to require little specialized knowledge and comparatively little effort to use. Further, these general tools tend to be non-interactive, and often have long compile times.

3 The Core Design of Vulcan

The core functionality of VULCAN is to visualize entries in a corpus, as well as model predictions on that corpus. Advanced functionality includes the display of further information such as alternative predictions and their likelihood. A full description of features follows in Section 4.

Fig. 1 shows an example, where VULCAN visualizes an entry from the Little Prince AMR corpus.[1] This is the third sentence in the corpus, displayed together with the annotated AMR from the corpus (left) and a predicted AMR (right; this AMR was predicted by the parser of Groschwitz et al., 2018).

Throughout this paper, we will make a distinction between the *host*, who runs VULCAN and provides the data, and the *viewer*, who looks at the visualizations. In practice, host and viewer can be the same person: take for example a researcher who uses VULCAN to visualize her model's predictions on the development set, in order to analyze

[3]https://demos.explosion.ai/displacy
[4]https://github.com/cbaziotis/neat-vision
[5]https://github.com/ablodge/amr-utils
[6]E.g. http://lindat.mff.cuni.cz/services/teitok/ud211/index.php?action=browser&class=lang and https://clarino.uib.no/iness-prod/sentences.

[7]https://tikz.net
[8]https://graphviz.org

errors and find the best next step for improving the model. This researcher is then both the host and the viewer. But if, say, VULCAN is used to host a visualization of a corpus on the web, to showcase the corpus to the public, then host and viewer are different people.

Following this principle of host and viewer, VULCAN uses a server/client design. The server side is operated by the host; the viewer interacts with the client side, which runs in the browser. Thus, the viewer simply sees an intuitive browser interface like the one in Fig. 1.

During setup by the host, VULCAN takes as input a file in a dictionary format (see Section 6), containing all (and only) the information required for the visualization. All model predictions, attention etc. that are to be visualized must be included in this input file, which uses a simple, generic format – this allows VULCAN to be agnostic about the model's implementation framework. The dictionary can be presented as a `pickle` or `JSON` file (generating `JSON` is particularly widely supported across programming languages). Inside the dictionary, structures like graphs and trees can be included in a variety of formats: VULCAN supports a range of input codecs (Section 6) that can be further extended (Section 7).

The visualization can then be accessed from a browser as long as the server program runs. Viewers can browse the corpus, and access further information through e.g. mouseover interaction (Sections 4, 6.2).

4 Features

This section describes the visualization capabilites of VULCAN in detail.

Throughout this section, we will use three examples:

`UD` Visualizes the PUD test set of the Japanese Universal Dependency treebank (Fig. 2; McDonald et al., 2013). This is an example of visualizing only a corpus and no model predictions. This also highlights VULCAN's ability to render characters outside the latin alphabet.

`LEAMR` Visualizes the alignments predicted by LEAMR (Blodgett and Schneider, 2021) on the Little Prince AMR dataset.[1] LEAMR aligns nodes of the AMR semantic graphs to tokens, using a combination of heuristics and machine learning.

`AM-parser` Visualizes the predictions of the AM parser (Groschwitz et al., 2018) on the Little Prince AMR dataset.[1] The AM parser is a neuro-symbolic compositional parser that predicts a so-called *AM tree* for a given sentence, which consists of graph fragment supertags and dependency edges that represent graph-combining operations. The AM tree then deterministically evaluates to an AMR graph. Here, VULCAN shows the AM tree both directly on top of the sentence and standalone (right), and shows both the gold and the predicted AMR for comparison (left and center respectively).

4.1 Visualized structures.

Vulcan can currently visualize:

- Strings and their tokenization (all examples),
- Tags for strings (`UD`, `AM-parser` examples, Figs. 2, 4),
- Tables,
- Trees (`AM-parser` example, Fig. 4),
- Dependency trees (`UD`, `AM-parser` examples, Figs. 2, 4),
- Graphs (`LEAMR`, `AM-parser` examples, Figs. 3, 4).

Each of these structures has labeled *elements*: tokens for sentences, cells in a table, nodes in a graph etc. Each such element can be labeled with a string, but also with a more complex structure from the list above. For example, note how in Fig. 4, the supertags on top of the string are themselves graphs, and the nodes in the AM tree on the right are labeled with graphs as well.

4.2 Alignments and model internals

VULCAN displays alignments by highlighting aligned elements in blue when mousing over an element. For example, in the `LEAMR` visualization, when the mouse hovers over a token in the sentence, aligned graph nodes are highlighted (and vice versa).

Similarly, attention (e.g. Bahdanau et al., 2015) is visualized: the higher the attention weight, the stronger the highlighting shade (Fig. 5). VULCAN can also visualize transformer's multihead attention (Vaswani et al., 2017). In that case, the highlighting takes the shape of a matrix, where the different heads of one layer are each represented in one row

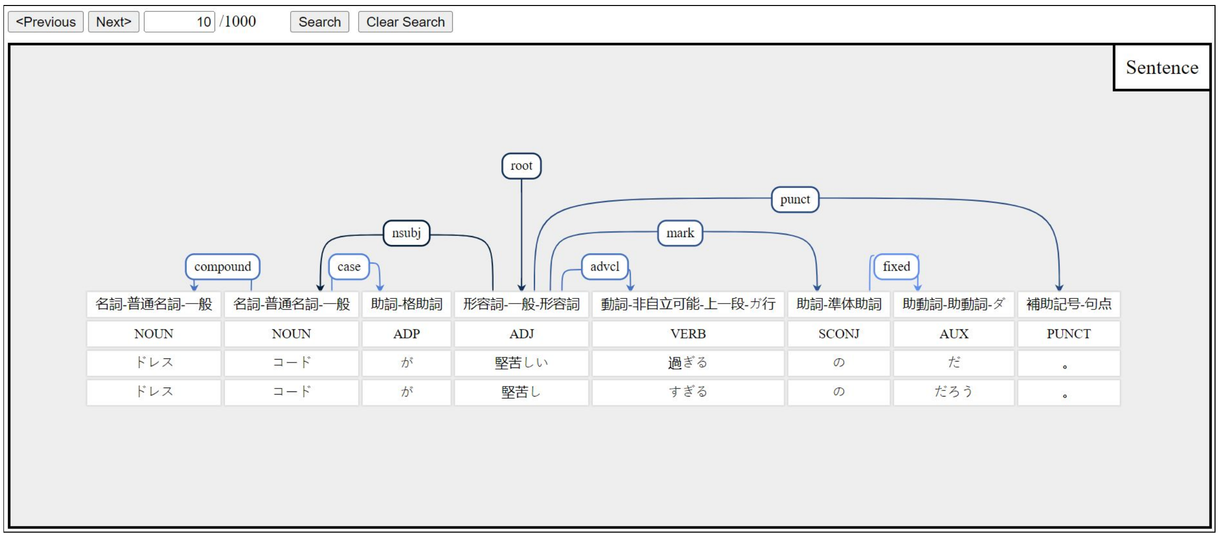


Figure 2: VULCAN visualizing a dependency tree over Japanese text (Universal Dependencies).

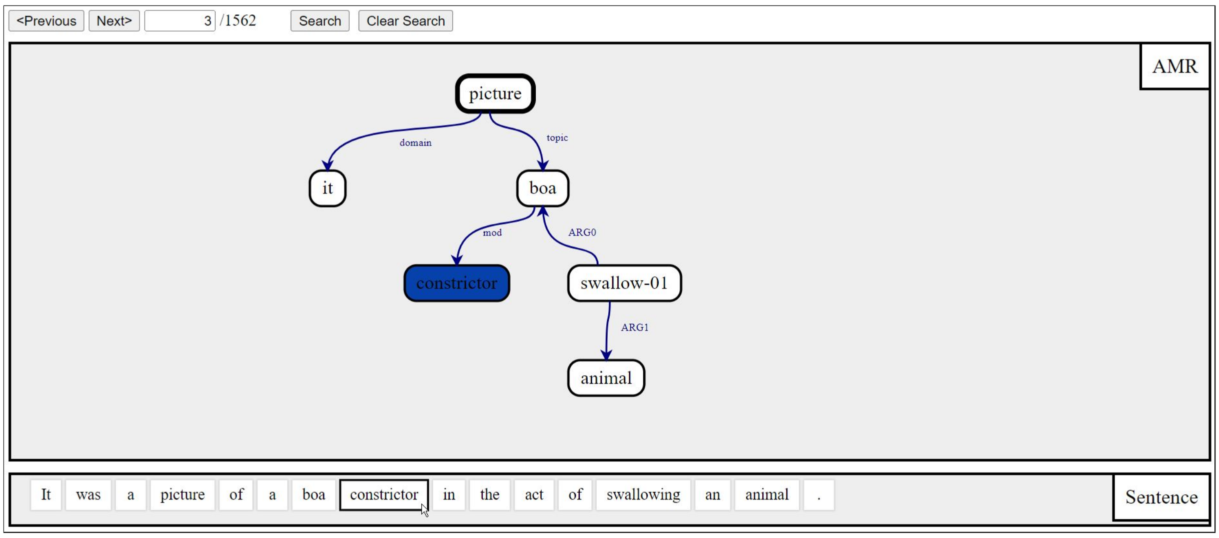


Figure 3: VULCAN visualizing alignments for an AMR from the Little Prince corpus. The mouseover on the token 'constrictor' highlights the aligned node in the graph.

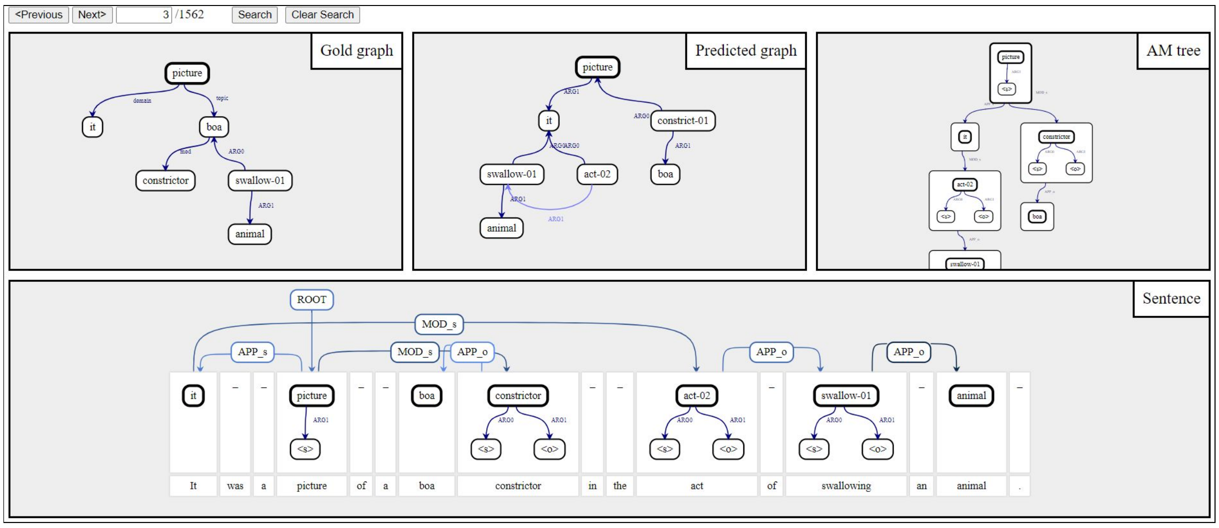


Figure 4: VULCAN visualizing predictions of the AM parser.

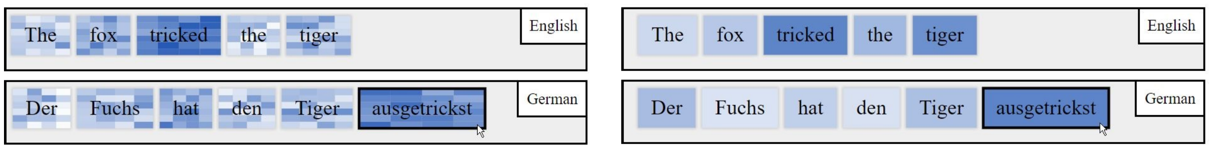


Figure 5: VULCAN visualizing multihead attention (left) and the attention of a single head (right), on a machine translation example.[9]

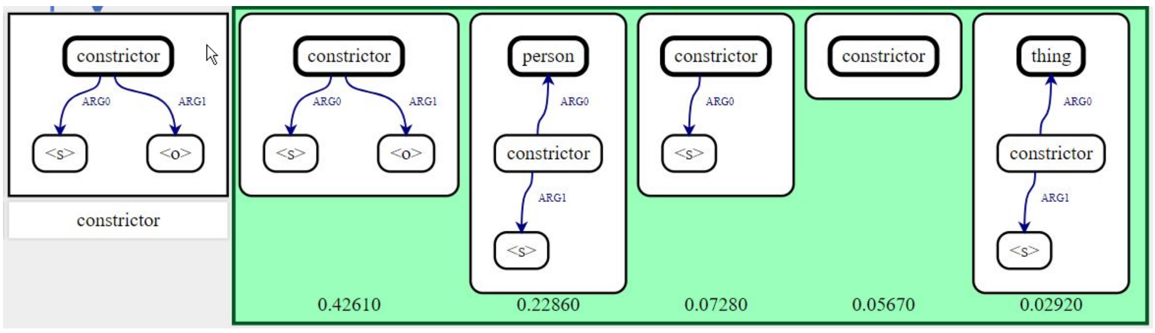


Figure 6: VULCAN visualizing alternative supertags for the token *constrictor* from the example shown in Fig. 4 (`CTRL` + mouseover effect; screenshot of detail).

(Fig. 5, left). The viewer can also select a specific attention head from a drop-down menu in the web-interface, to show only the weights from that head (Fig. 5, right).

Finally, for each element, VULCAN can display alternative labels predicted by the model, and their assigned probabilities, as long as this information is provided by the host. This feature is accessed by mousing over an element with the `CTRL` key pressed. Fig. 6 shows the top five supertags predicted as most likely, for the `AM-parser` example.

4.3 Search function

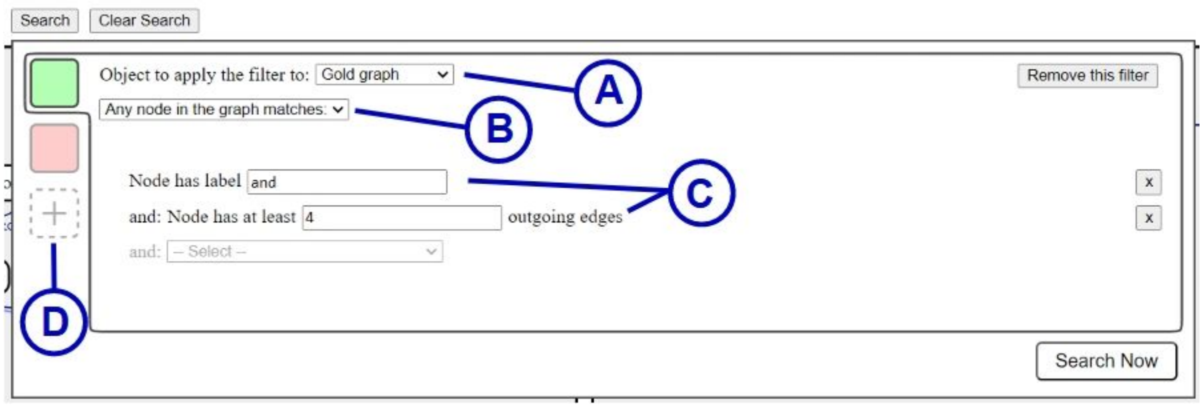
VULCAN provides a search interface to filter a corpus for entries matching certain criteria. The search functionality is based on a range of predefined search patterns that can also be combined. An example search for the visualization shown in Fig. 1 is illustrated in Figs. 7-9.

Any search can have multiple *search filters* that a corpus entry must all satisfy in order for it to be included in the search results. The two search filters of this example are shown in Fig. 7 and Fig. 8.

For each search filter, the viewer first selects which structure of the corpus entry the filter applies to (Fig. 7A). Then, an *outer search pattern* (Fig. 7B) is selected. For example, should the filter focus on the graph's nodes, on the edges, or on the graph as a whole?

Then, multiple *inner search patterns* (Fig. 7C)

[9]Note that this example is fictional, for illustrating VULCAN's capabilities only. No actual model was trained.


Object to apply the filter to: Gold graph

Any node in the graph matches:

Node has label and

and: Node has at least 4 outgoing edges

and: – Select –

Figure 7: VULCAN's search interface (screenshot of detail). The blue markers A-D were added to the screenshot for reference in the paper.

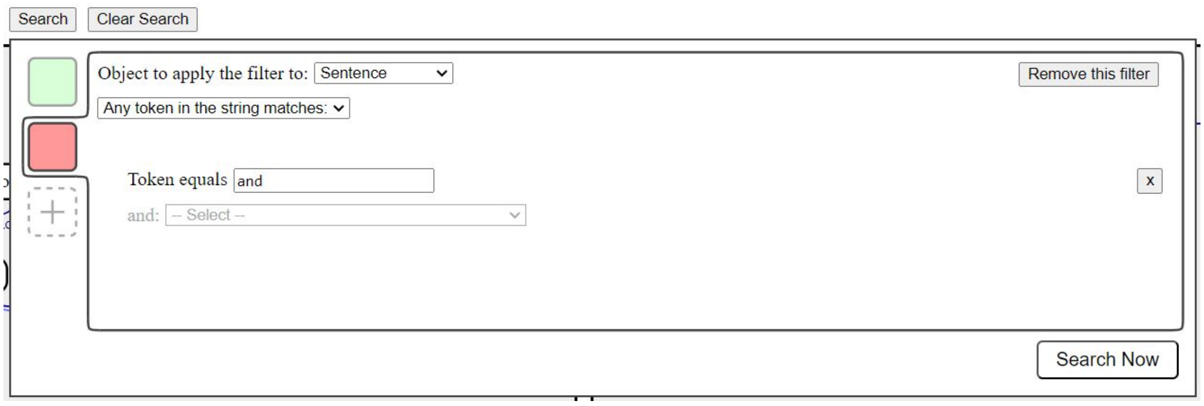


Figure 8: Second search filter used in Fig. 9 (screenshot of detail).

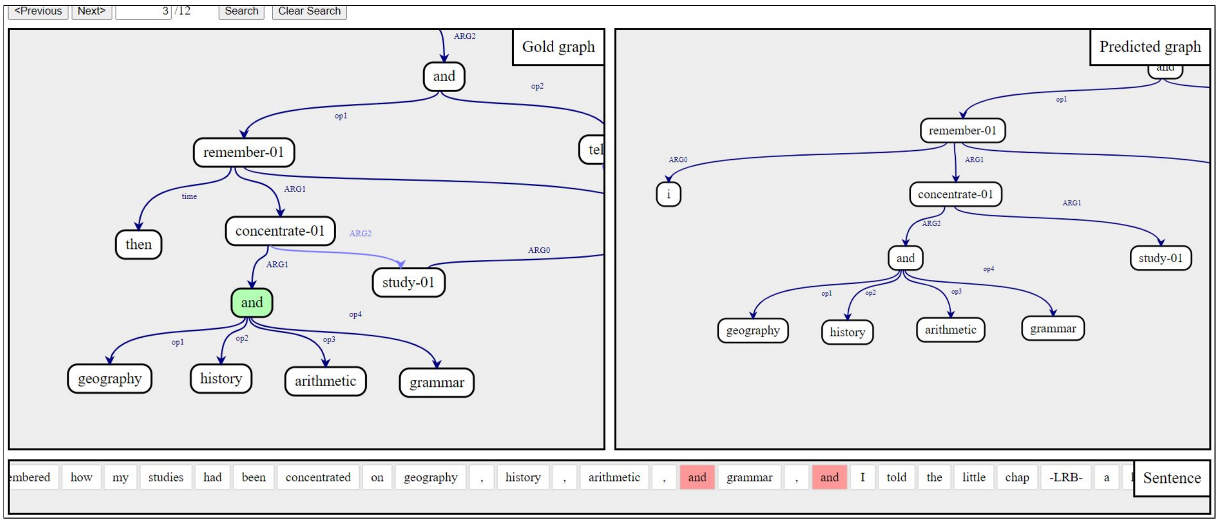


Figure 9: Search results with highlights. The search filters used here are the ones from Figs. 7 and 8, with the matching nodes and tokens highlighted in green and red respectively.

can be selected (the available inner search patterns depend on the selected outer pattern). The inner patterns specify the actual search criteria, e.g. a node having a certain label.

Executing the search then displays all corpus entries that match all patterns. The structure's elements (here nodes and tokens) that match the patterns are highlighted in the search results, see Fig. 9. Each filter has its own color (Fig. 7D), here green and red.

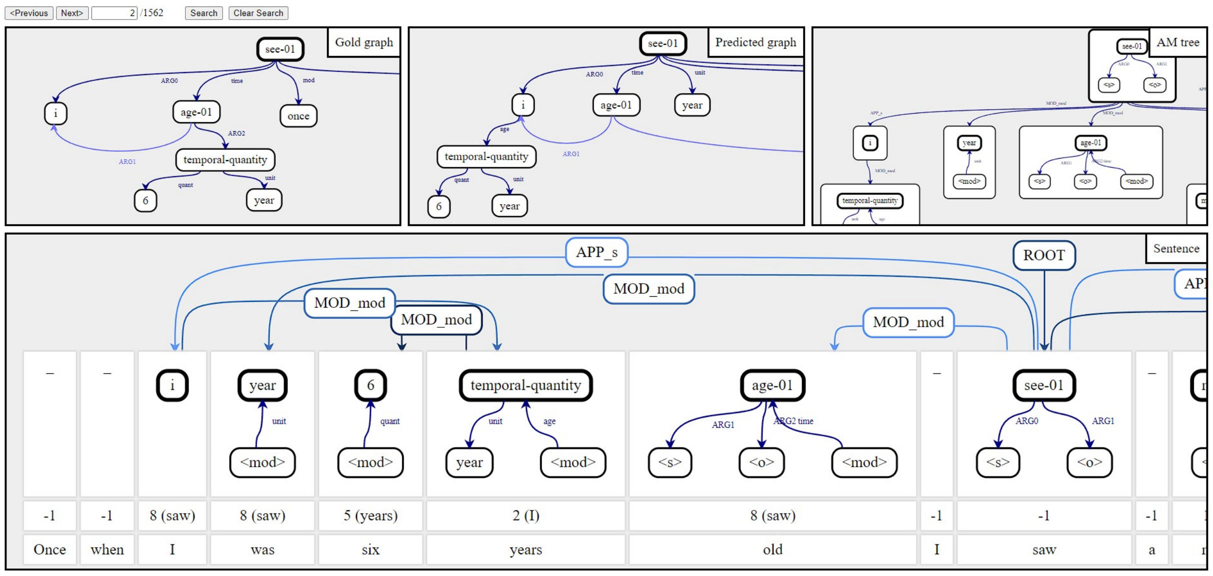
5 Case Study

To illustrate the kinds of insights VULCAN makes available, let us take a closer look at the `AM-parser` example from Section 4. Specifically, we will investigate the sentence *Once when I was six years old I saw a magnificent picture in a book, called True Stories of Nature, about the primeval forest.* The VULCAN visualization is shown in Fig. 10. In this case study, we will gain insights into the probability distributions that the parser predicts in practice and find the reasons why the parser makes a specific error. Based on these insights, we develop concrete ideas for a statistical analysis to confirm the found issues, and possible model changes to address them.

But before we jump into the analysis, let us look at the AM parsing model in more detail (see Groschwitz et al., 2018 for a full description). Recall that the AM parser predicts a so-called AM tree, consisting of dependency edges and supertags, that then evaluates to an AMR graph. Given a sentence, the parser computes for each token, let us say here the token at position i, three types of scores:

1. The *supertag scores* $\omega(G)$ for each possible graph fragment supertag G from a lexicon during training.[10]

[10]Technically, this score is further split into scores for *delex-*



_	_	i	year	6	temporal-quantity	age-01	_	see-01	_
-1	-1	8 (saw)	8 (saw)	5 (years)	2 (I)	8 (saw)	-1	-1	-1
Once	when	I	was	six	years	old	I	saw	a

Figure 10: Vulcan visualization for the sentence in the case study of Section 5, with each structure zoomed in on the relevant parts. From left to right, top to bottom: gold AMR graph, predicted AMR graph, predicted AM tree, input sentence with predicted AM tree. The numbers above each token specify the head of that token; -1 signifies a dummy token outside the sentence for tokens that do not contribute to the actual AM tree.

2. The *head scores* $\omega(k \rightarrow i)$, the likelihood that the head of the token at position i is the token at position k. I.e. the likelihood that the incoming edge at position i comes from position k.

3. The *edge label scores* $\omega(l|k \rightarrow i)$ that a dependency edge from k to i has label l.

A symbolic decoding algorithm then finds the best AM tree based on these scores and the type constraints of the AM algebra. This AM tree then evaluates deterministically to the graph.

In the visualization in Fig. 10, the heads and supertags of that best AM tree are shown above each token, and the dependency edge labels (`APP_s` etc.) are shown directly on the edge. Note that some tokens in the graph do not have a supertag assigned to them, and are not part of the dependency trees. These tokens do not contribute directly to the final graph, and the parser has decided to ignore them. For a token to be *ignored* this way, three things must happen: (1) it must have the dummy supertag represented here with an underscore "_", (2) its head must be a ficticious token at position -1, and (3) the label of the incoming edge must be `IGNORE` (note that `IGNORE` edges are omitted in the visualization here). That is, the supertag, head and edge label scores all contribute to a token being ignored (they do not have to all say that ignoring the token is most likely, but in total must make ignoring the token the most likely choice for the decoding algorithm).

icalized supertags and separate lexical labels. This detail is however not relevant for the analysis here, and we bundle the scores into one.

This case study focuses on the token *was*, the fourth token in the sentence. It should be ignored – it does not directly contribute information that is represented in the gold AMR –, but it is not. Here we want to find out why.

Figures 11-13 show the top five supertags, heads and incoming-edge-labels, respectively, for the token *was*. Each possible prediction is given with its score ω as a probability. We can make a first observation immediately: the scores differ wildly in how confident the model is. The scores for the supertag (Fig. 11) have medium confidence, with the dummy supertag "_" having the highest score by quite a margin, but the other options also having a significant amount of probability mass among them. By contrast, the score distribution for the head (Fig. 12) is very flat. The option of choosing *saw* as the head, (wrongly) predicted as most likely, has just barely a higher score than the (correct) choice of the dummy token at position -1, which is fourth most likely. By contrast in the other direction, the distribution for the edge label is extremely peaky, assigning nearly all of the probably mass to the label `MOD_mod`. The correct edge label `IGNORE` is not even in the top five.

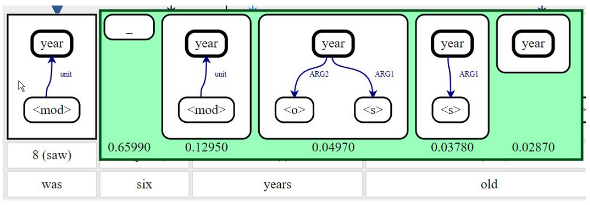


Figure 11: Top five supertags with probability scores for the token *was* (CTRL + mouseover effect).

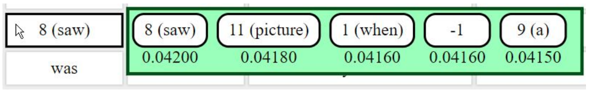


Figure 12: Top five possible heads with probability scores for the token *was* (CTRL + mouseover effect).

This explains why the token *was* is not ignored: while the supertag and head prediction together would indicate ignoring the token, they are not strong enough to overcome the confidently wrong edge label prediction.

A first takeaway then is that the score distributions may be unevenly balanced in general. Looking at some more examples further supported this hypothesis. A possible next step would be to examine the "peakiness" and "flatness" of the score distributions with corpus-wide metrics, for example by measuring the average entropy of supertag, head, and edge label scores, and see if the pattern holds: that the head scores are consistently overly flat, and edge label scores are consistently too peaky. If so, countermeasures such as hyperparameters to balance out the scores during decoding would be a promising avenue to improve parser performance.

A second takeaway is to look more closely at the edge label predictions. Why are they so confidently wrong? The answer lies in the following design decision of the AM parser. To save computation time, not all edge label scores $\omega(l|k \to i)$ are computed. Instead, for each token position i, edge label scores are only computed for the most likely head k_{max}, and then this distribution is used for all k. That is, for all possible heads k, the score $\omega(l|k \to i)$ is approximated as $\omega(l|k_{\text{max}} \to i)$. In our example, $k_{\text{max}} = 8$ (*saw*). But the training data taught the parser that IGNORE edge labels go with the dummy head $k = -1$. Thus, the score of the IGNORE label here, which would be the correct prediction, is very low. As a conclusion, this shortcut to save computation time has a cost in terms of accuracy here. Revisiting this decision is another promising

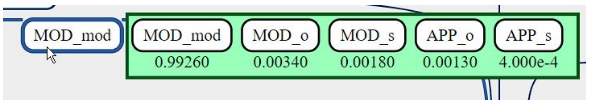


Figure 13: Top five edge label alternatives with probability scores for the incoming edge to the token *was* (CTRL + mouseover effect).

avenue to improve parsing performance.

This concludes the case study. Looking at the probability distributions for just a single token generated multiple hypotheses for how the parser could be improved. A good first next step towards confirming these hypotheses would be to look at more examples and see if the pattern holds. Then, corpus-wide statistical analyses can serve as further confirmation where applicable. Finally, implementing model changes, and iterating this pattern of evaluation, interpretation and implementation continue the research loop.

6 Usage and Data Formats

6.1 Hosting corpora with VULCAN

The key step in hosting a corpus with VULCAN is to create a *VULCAN visualization file*. Once such a file exists, running vulcan takes just one line (technical details in the code documentation).

Technically speaking, a VULCAN visualization file is simply a dictionary in a specific format, which can be provided as a JSON or pickle file. However, VULCAN contains Python functions that allow building a VULCAN visualization file in an intuitive way, without having to worry about the concrete dictionary format.

The general idea is that each entry in the corpus has a fixed set of structures that we want to visualize. For example, in Fig. 1, there are structures named the ‘Gold graph’, ‘Predicted graph’ and ‘Sentence’. To build a VULCAN visualization file, one first specifies this fixed list of structures that each corpus entry consists of. Then one adds the corpus entries one by one.

More technically speaking, the first step is to create a VulcanFileBuilder object, whose constructor takes as only parameter a dictionary like this (again, for the example in Fig. 1):

```
{"Gold graph":      "penman_string",
 "Predicted graph": "penman_string",
 "Sentence":        "string"}
```

It maps structure names (like “Gold graph”) to their *visualization type* (like “penman_string“). The visualization type is a string that describes what input codec to use for a structure, and how the structure will be visualized (as a table, graph, etc.). For example, “penman_string“ is registered in VULCAN as a format that uses the Penman[11] input codec to read a string encoding of a graph, and then displays the output as a graph. VULCAN features a range of input codecs for common formats, such as Penman graphs and NLTK[12] trees.

After this initialization, corpus entries can be added with `VulcanFileBuilder`'s `add_instances_by_name` function. The function adds one corpus entry at a time, and takes as only argument a dictionary like this:

```
{"Gold graph":      "(b / be-loc...",
 "Predicted graph": "(b / be-loc...",
 "Sentence":        "Here is a copy of
                        the drawing ."}
```

(with the string encodings of the graphs abbreviated here). The dictionary maps structure names to the actual structures for this corpus entry, encoded in a way that is compatible with the input codec specified during initialization. For example, the string “`(b / be-located-at-91 :ARG1 ...`” (and so on) is in the correct format for the Penman codec.

Alignments, attention scores and alternative label predictions can be added similarly, using the following format. In VULCAN, each element within a structure (a token in a string, a cell in a table, a node in a graph, etc.) has a unique identifier in that structure, the *element name*. For example, a token's name is its position in the sentence. Alignments and attention are specified as dictionaries that map pairs of element names to a score (for alignments, the scores are 1 or 0). Alternative label predictions are also encoded as a dictionary, mapping an element's name to a list of its alternative labels (and their scores). These generic formats make it easy to create input for VULCAN from any source.

A VULCAN visualization file can also be built by hand, e.g. if the host prefers building it in a programming language other than Python. The full dictionary file format is described in Appendix A.

6.2 Viewing corpora with VULCAN

Viewing a VULCAN visualization essentially consists of opening the respective website in a browser. Navigating and searching the corpus is performed with self-explanatory buttons. Intuitive zoom (mousewheel) and drag gestures are implemented so that the viewer can focus in on a specific part of an object (like in Fig. 10), or view the object as a whole. Mousing over an element shows corresponding alignments and attention, where applicable (Figs. 3, 5). Mousing over an element while holding the `CTRL` key displays alternative label predictions (if given; e.g. Fig. 6).

6.3 Under the hood

The server side of VULCAN is implemented in Python. Communication between the client and the server uses Eventlet[13] and Socket.IO.[14] VULCAN can be hosted both locally and on the web. The client side is implemented in JavaScript and D3.[15]

7 Extensibility

VULCAN is modular, built with extensibility in mind. In particular, it is straightforward to add new input codecs, and to add new search patterns. This way, the host can customize VULCAN to their needs.

Future work could also add functionality to visualize completely new objects, such as images, or the ability to play sound files.

8 Conclusion

We have displayed the capabilities of VULCAN, a visualization tool for linguistic structures, neural models, and their interactions.

Among the features showcased in this paper are the visualization of strings, trees, graphs and complex supertags, as well as alternative predictions with their likelihoods and multi-head attention. This combination of features makes VULCAN particularly suited to facilitate the interpretation of neuro-symbolic models in practice. VULCAN also features a powerful search functionality and has broad compatibility due to its flexible input format.

We built VULCAN to make it easier to gain insights into corpora and models by looking at examples. We hope that VULCAN will play a part in making neural models in natural language processing and generation more interpretable and tangible.

[11] https://github.com/goodmami/penman

[12] https://www.nltk.org/

[13] https://eventlet.net/

[14] https://socket.io/

[15] https://d3js.org/

Limitations

One potential application of VULCAN is formal error analysis. However, in this use case, VULCAN can currently only contribute the visualization. Other tasks, such as logging of errors, must be done separately (though possible future work could extend VULCAN to allow error annotation within the visualization interface). For a detailed discussion of good practices in error analysis, see e.g. van Miltenburg et al. (2021).

As mentioned in Section 7, VULCAN does not yet support visualization of multimodal elements such as images, audio or video.

Ethics Statement

We do not see any particular ethics concerns with this work.

Acknowledgements

Many thanks go to Meaghan Fowlie and Matthias Lindemann for their feedback and encouragement. Special thanks to Ivan Titov and Raquel Fernández for their support in making this work possible. This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 492792184.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Austin Blodgett and Nathan Schneider. 2021. Probabilistic, structure-aware algorithms for improved variety, accuracy, and coverage of AMR alignments. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3310–3321, Online. Association for Computational Linguistics.

Ann L Brown and Mary Jo Kane. 1988. Preschool children can learn to transfer: Learning to learn and learning from example. *Cognitive psychology*, 20(4):493–523.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Mary L Gick and Keith J Holyoak. 1983. Schema induction and analogical transfer. *Cognitive psychology*, 15(1):1–38.

Jonas Groschwitz, Matthias Lindemann, Meaghan Fowlie, Mark Johnson, and Alexander Koller. 2018. AMR dependency parsing with a typed semantic algebra. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1831–1841, Melbourne, Australia. Association for Computational Linguistics.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal Dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria. Association for Computational Linguistics.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: an evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Michael I Posner and Steven W Keele. 1968. On the genesis of abstract ideas. *Journal of experimental psychology*, 77(3p1):353.

František Trebuňa and Ondrej Dusek. 2023. VisuaLLM: Easy web-based visualization for neural language generation. In *Proceedings of the 16th International Natural Language Generation Conference: System Demonstrations*, pages 6–8, Prague, Czechia. Association for Computational Linguistics.

Emiel van Miltenburg, Miruna Clinciu, Ondřej Dušek, Dimitra Gkatzia, Stephanie Inglis, Leo Leppänen, Saad Mahamood, Emma Manning, Stephanie Schoch, Craig Thomson, and Luou Wen. 2021. Underreporting of errors in NLG output, and what to do about it.

In *Proceedings of the 14th International Conference on Natural Language Generation*, pages 140–153, Aberdeen, Scotland, UK. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Eric Wallace, Jens Tuyls, Junlin Wang, Sanjay Subramanian, Matt Gardner, and Sameer Singh. 2019. AllenNLP interpret: A framework for explaining predictions of NLP models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 7–12, Hong Kong, China. Association for Computational Linguistics.

A Format of VULCAN visualization files

VULCAN visualization files, the input to the VULCAN software, consist of basic list and dictionary structures. VULCAN can read them in `pickle` and `JSON` format.

At the top level, VULCAN visualization files are lists of dictionaries. The list contains two types of dictionaries: one for *data* that specifies the structures (graphs, strings, etc.) to be visualized. and one for *linkers* that specifies alignments and attention.

A *data* dictionary describes all structures of one type in the corpus. For example, for the visualization in Fig. 1, there is one data dictionary for all "Gold graph" objects, one for all "Predicted graph" objects, and one for all "Sentence" objects. A data dictionary has the entries specified in Table 1:

A *linker* dictionary describes a relation between two structures: alignments or attention (both use the same format). It has entries as specified in Table 2

A.1 Label alternatives format

The entries for one 'label alternative' are:

Dictionary key	Value
"label"	The alternatively predicted label (can itself be a string, graph, etc).
"format"	Like the format in the data dictionary above. Specifies the format the label is in.
"score"	The score (or likelihood) that was predicted for this label

We recommend specifying the top k label alternatives, where k is around 3-5.

Key	Value
“type”	For data dictionaries, this is always the string “data”
“name”	The name of this structure, e.g. “Gold graph”. This is displayed in the top right of the visualization for this structure.
“format”	A name from a predefined list of possible formats. This specifies the format in which the structures are provided; in particular, what input codec to use and whether to display the structure as a string, graph, etc.
“instances”	The list of objects of this type in the corpus. For example, all gold graphs, or all predicted graphs, or all sentences. The objects in this list must be in the format specified in the “format” entry.
“label_alternatives”	Optional. A list, with one entry for each instance in “instances”. Each such entry is a dictionary, mapping element names to a list of ‘label alternatives’. Each ‘label alternative’ is itself a dictionary with entries specified in Section A.1.
“dependency_trees”	Optional. Only available if the structures are strings (or tagged strings, which are treated as tables on a technical level). A list of dependency trees, one for each entry in “instances”. A dependency tree is a list of triples (source, target, label), where source is the 0-based index of the edge’s origin (-1 if the edge has no origin, e.g. the root indicator), target is the 0-based index of the word the edge points to, and label is the edge label (a string).

Table 1: Key-value pairs of the data dictionary.

Dictionary key	Value
“type”	For linker dictionaries, this is always the string “linker”
“name1”	The name of one of the structures that this links (for example, "AMR" in Fig. 3).
“name2”	The name of the other structure that this links (for example, "Sentence" in Fig. 3).
“scores”	A list of *outer dictionaries*, one for each entry in the corpus. Each outer dictionary maps element names from the structure with name *name1* to *inner dictionaries*. Each inner dictionary maps element names from the structure with name *name2* to scores between 0 and 1. Closer to 1 means a higher attention weight. For alignments, use 1 for aligned (and 0 for not aligned, but this is the default value and does not need to be specified). In other words, the outer and inner dictionaries describe a sparse matrix between the element names of the structure *name1* and the element names of the structure *name2*; the entries in the matrix are the scores. (Sparse in the sense that not all entries in the matrix need to be specified in the matrix; they default to 0).

Table 2: Key-value pairs of the linker dictionary.