Appendix of the paper "From document to program embeddings: can distributional hypothesis really be used on programming languages?"

Thibaut Martinet, Guillaume Cleuziou, Matthieu Exbrayat, Frédéric Flouvat

Appendix of the paper "From document to program embeddings: can distributional hypothesis really be used on programming languages?"

Thibaut Martinet[a,*], **Guillaume Cleuziou**[a], **Matthieu Exbrayat**[a] **and Frédéric Flouvat**[b]

[a]University of Orléans, INSA-CVL, LIFO, EA 4022, F45067 Orléans, France
[b]Aix Marseille Univ, CNRS, LIS, Marseille, France
ORCID (Thibaut Martinet): https://orcid.org/0000-0002-0170-9260, ORCID (Guillaume Cleuziou): https://orcid.org/0000-0002-2885-1152, ORCID (Matthieu Exbrayat): https://orcid.org/0000-0002-1740-4752, ORCID (Frédéric Flouvat): https://orcid.org/0000-0001-7288-0498

A Implementation

In order to compute the different program representations, we use the python library tree-sitter[1], which contains parsers for a number of programming languages, and allows us to parse a piece of code and compute its AST, with leaf nodes corresponding to every token of the code, and some additional abstract nodes to represent the hierarchical structure of code.

About our AST format, we chose to convert tree-sitter one into a custom one, in which the nodes correspond to so-called instructions, which are either simple expressions (*e.g.* assignment, function call, *etc*), control structure headers (without body statements, *e.g.* if/else, loop, *etc*), or declaration statements (*e.g.* function, class, *etc*). Each node has a list attribute containing the corresponding tokens. This way, we can extract the set of instructions as well as the set of tokens, reconstructing the original source code by walking through the nodes in a depth-first order.

A.1 Anonymization of source code

First of all, we anonymized comments by replacing each one by a token *_COMMENT_*. We also anonymized each declared function and class name by a token *_FUNCTION_NAME_* or *_CLASS_NAME_* respectively. Next, we anonymized each imported module and tools by a token *_MODULE_* or *_TOOL_* respectively. And we anonymized every constant by a token containing its type, *e.g.* *_CONST_INT_*, *_CONST_FLOAT_*, *etc*.

Finally, we tried different anonymization strategies for the variables and function parameters, since they are the most impacting element of the code. We replaced the variables by a token *_VAR_*, but we tried either to anonymize with or without a number, *i.e.* *_VAR1_*, *_VAR2_* and so on. We tried the same thing with function parameters, but also either by a token *_VAR_* or a token *_PARAM_*. We ended up with the best results when anonymizing both with *_VAR_* and without numbers. Here is an example of the resulting anonymization strategy of the program below:

```
0  def _FUNCTION_NAME_(_VAR_):
1      _VAR_ = _CONST_NONE_
2      for _VAR_ in _VAR_:
3          if _VAR_ is _CONST_NONE_ or _VAR_ > _VAR_:
4              _VAR_ = _VAR_
5      return _VAR_
```

B Program representation examples

Here is an example of a program, corresponding to the anonymized one above, and to the ASTs and execution trace below:

* Corresponding Author. Email: thibaut.martinet@univ-orleans.fr

```
0  def max(l):
1      res = None
2      for e in l:
3          if res is None or e > res:
4              res = e
5      return res
```

B.1 AST example

Example of token-based and instruction-based ASTs are shown in figs. 1 and 2 respectively, based on the above source code.

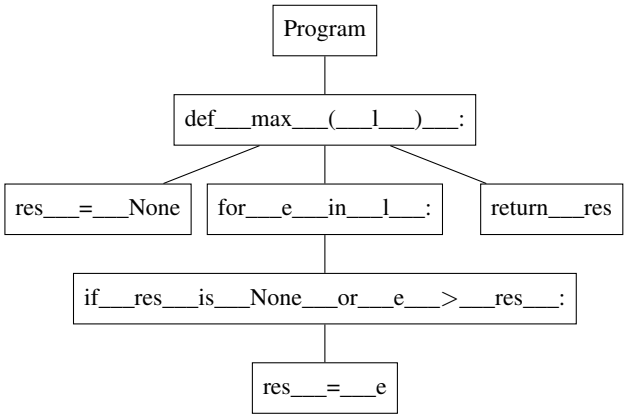

Figure 1. instruction-based AST

B.2 Execution trace example

Here is an example of execution trace, corresponding to the above source code executed with the input $[2, 3, 1, 0]$.

```
0  def max(l):
1  res = None
2  for e in l:
3  if res is None or e > res:
4  res = e
2  for e in l:
3  if res is None or e > res:
4  res = e
2  for e in l:
3  if res is None or e > res:
2  for e in l:
3  if res is None or e > res:
5  return res
```

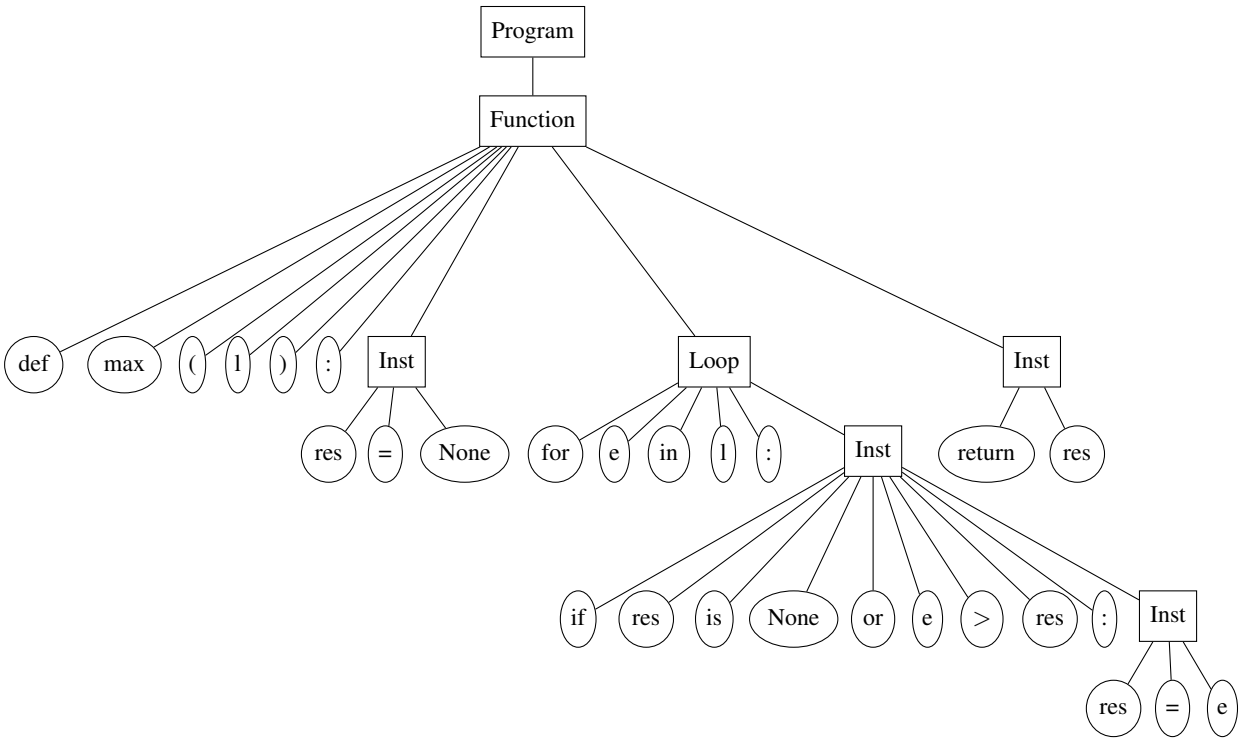

Figure 2. token-based AST

Table 1. Analogy evaluation per types, with PV-SG architecture, after 500 epochs on NC5690

	Source		Trace		AST	
	$Sec2vec^*(\mathcal{H}^t_S)$	$Sec2vec^*(\mathcal{H}^i_S)$	$Sec2vec^*(\mathcal{H}^t_T)$	$Sec2vec^*(\mathcal{H}^i_T)$	$Sec2vec^*(\mathcal{H}^t_A)$	$Sec2vec^*(\mathcal{H}^i_A)$
Syntactic analogies	0.335 ±0.016	0.203 ±0.011	0.268 ±0.015	0.164 ±0.010	**0.471** ±0.015	0.151 ±0.023
Semantic analogies	**0.966** ±0.000	0.006 ±0.003	0.964 ±0.001	0.007 ±0.003	0.942 ±0.012	0.007 ±0.003

C Additional evaluations with PV-SG and instruction-based models

The results of our framework on PV-SG architecture are shown in figs. 3 and 4 and table 1, and on instruction-based models in figs. 5 to 7.

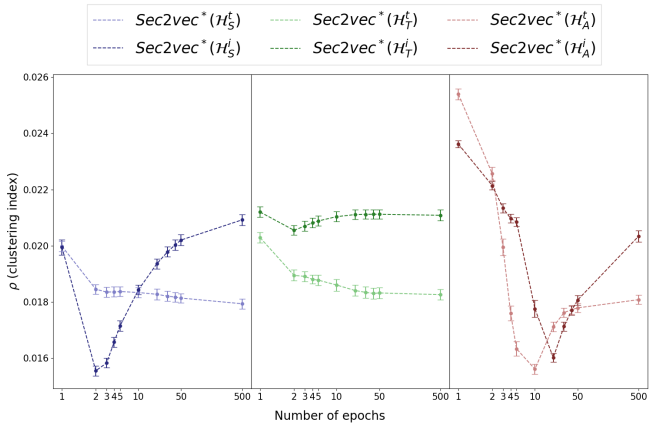

Figure 3. Clustering index ρ monitoring on dataset NC5690 the external information (here the exercises) captured by different sec2vec models (with PV-SG architecture): on source codes (blue), execution traces (green) and ASTs (brown).

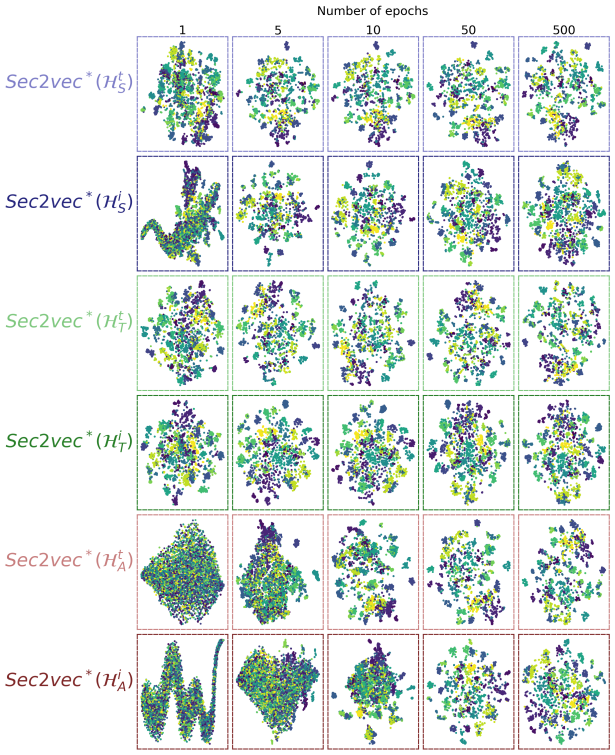

Figure 4. Cartography evolution on NC5690 from different distributional hypotheses, using the PV-SG architecture, after training phases of different number of epochs, reduced to 2 dimensions by t-SNE models. The dot colors are based on the exercises.

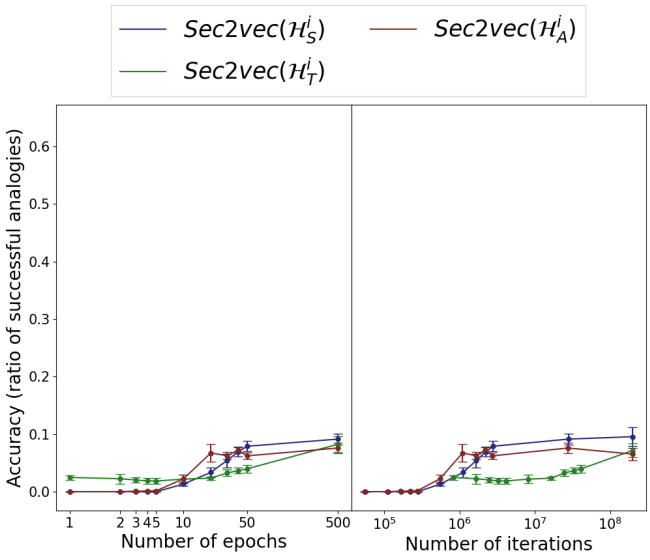

Figure 5. Analogy evaluation of different instruction-based sec2vec models applied on NC5690, after training phases of different number of epochs, aligned by number of epochs on the left, and by number of iterations on the right (one epoch consists of as many iterations as learning examples). The scale from the paper has been kept here.

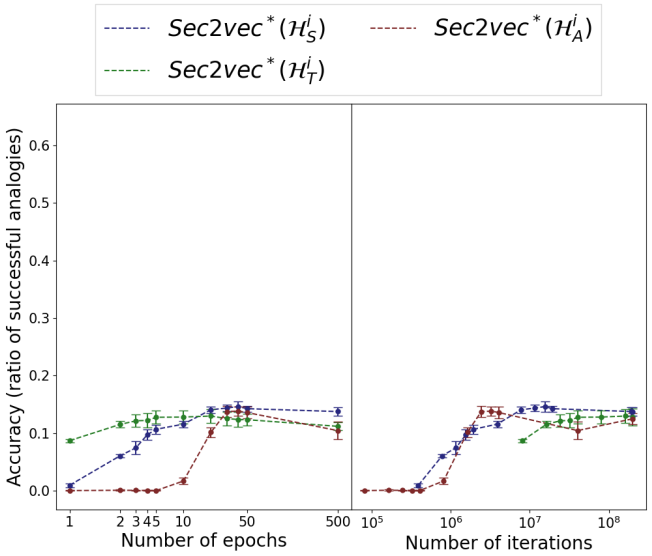

Figure 6. Analogy evaluation of different instruction-based sec2vec models (with PV-SG architecture) applied on NC5690, after training phases of different number of epochs, aligned by number of epochs on the left, and by number of iterations on the right (one epoch consists of as many iterations as learning examples). The scale from the paper has been kept here.

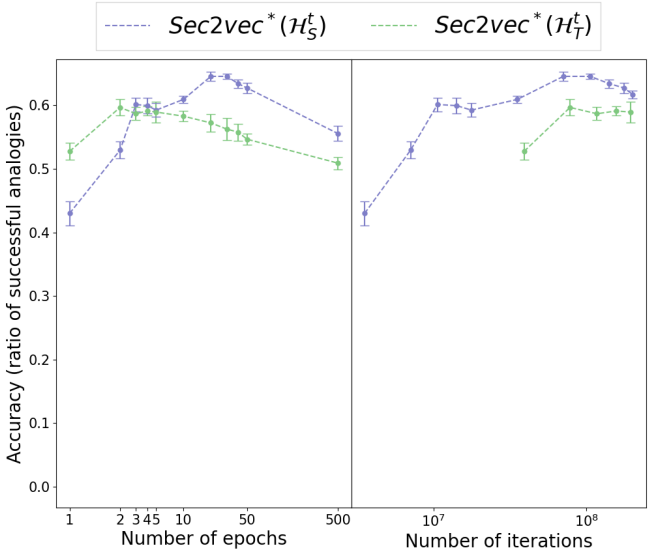

Figure 7. Analogy evaluation of sec2vec models (with PV-SG architecture) based on tokens in source code and execution trace, applied on NC5690, after training phases of different number of epochs, aligned by number of epochs on the left, and by number of iterations on the right (one epoch consists of as many iterations as learning examples). The scale from the paper has been kept here.

D Additional evaluations on other datasets

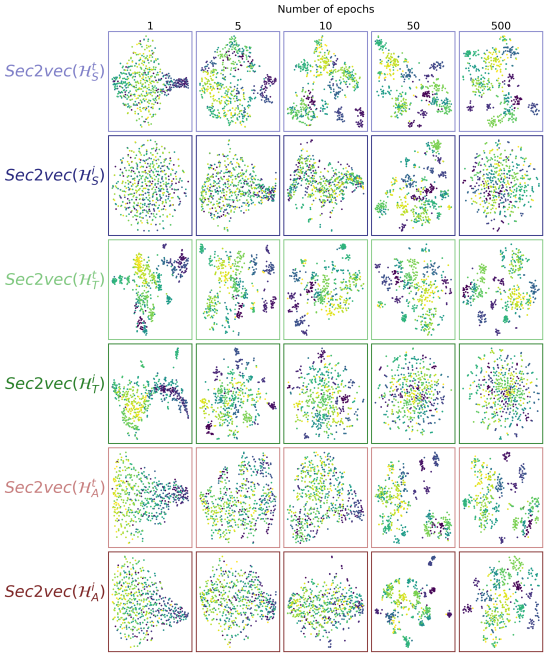

Figure 8. Cartography evolution on AD2022 (python) from different distributional hypotheses, after training phases of different number of epochs, reduced to 2 dimensions by t-SNE models. The dot colors are based on the exercises.

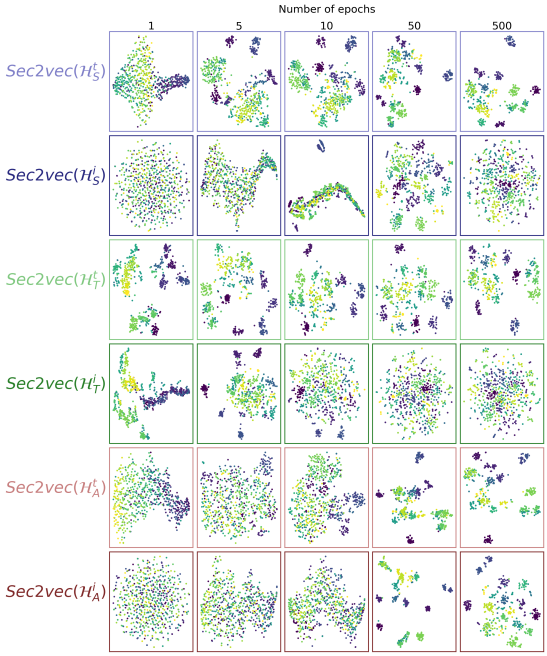

Figure 10. Cartography evolution on AD2022 (java) from different distributional hypotheses, after training phases of different number of epochs, reduced to 2 dimensions by t-SNE models. The dot colors are based on the exercises.

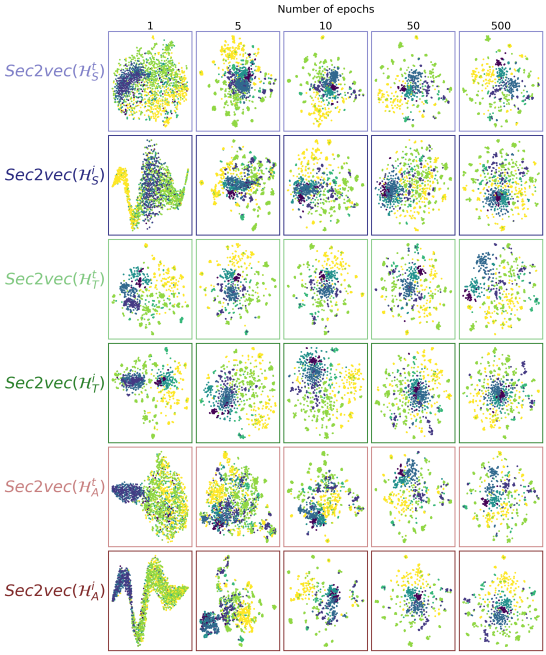

Figure 9. Cartography evolution on ProgPedia (python) from different distributional hypotheses, after training phases of different number of epochs, reduced to 2 dimensions by t-SNE models. The dot colors are based on the exercises.

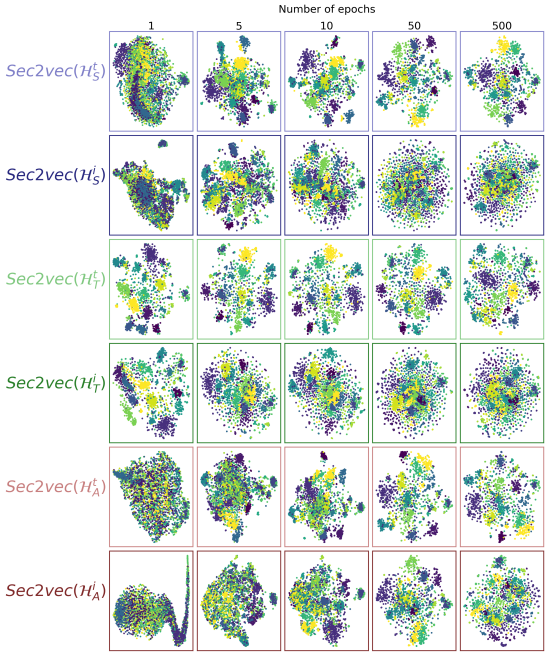

Figure 11. Cartography evolution on ProgPedia (java) from different distributional hypotheses, after training phases of different number of epochs, reduced to 2 dimensions by t-SNE models. The dot colors are based on the exercises.

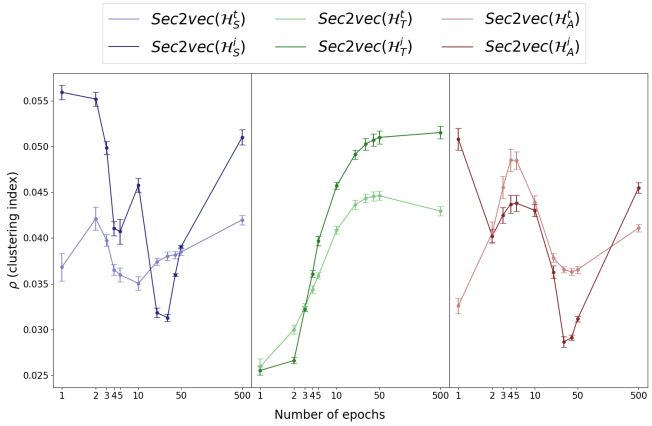

Figure 12. Clustering index ρ monitoring on dataset AD2022 (python) the external information (here the exercises) captured by different sec2vec models : on source codes (blue), execution traces (green) and ASTs (brown).

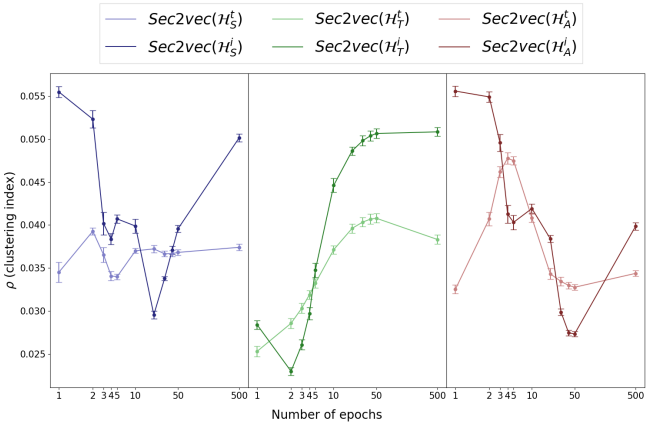

Figure 14. Clustering index ρ monitoring on dataset AD2022 (java) the external information (here the exercises) captured by different sec2vec models : on source codes (blue), execution traces (green) and ASTs (brown).

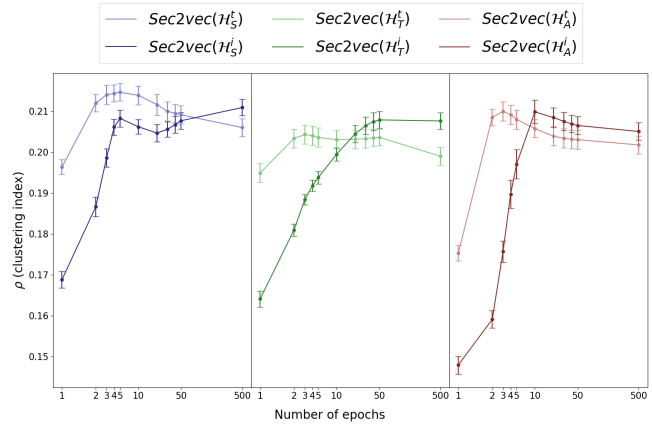

Figure 13. Clustering index ρ monitoring on dataset ProgPedia (python) the external information (here the exercises) captured by different sec2vec models : on source codes (blue), execution traces (green) and ASTs (brown).

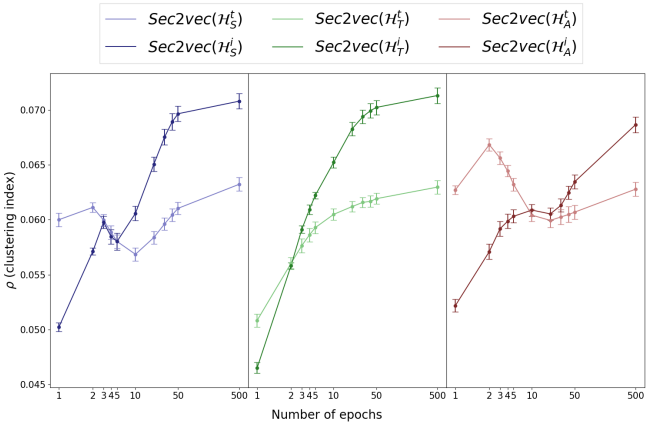

Figure 15. Clustering index ρ monitoring on dataset ProgPedia (java) the external information (here the exercises) captured by different sec2vec models : on source codes (blue), execution traces (green) and ASTs (brown).

References

[1] M. Brunsfeld, P. Thomson, J. Vera, A. Hlynskyi, P. Turnbull, T. Clem, and A. Muller. tree-sitter/tree-sitter: v0. 20.0, 2018.