

**Dear Editors and Reviewers,**

We would like to thank you for your effort invested in reviewing our submission and providing constructive feedback. This is much appreciated.

Below, we provide detailed answers (in green italics) to the points raised by the Reviewers (regular font). In the revised version, we did our best to address your remarks while keeping the article within reasonable length. Please notice that our original submission was already 16 pages long (not counting the appendices and bibliography), which, given the rather capacious template of the NAI Journal, puts quite a heavy burden on the reader.

As both Reviewers were pointing primarily to the incompleteness of the presentation of the DSL and related topics (program generation, program normalization), the majority of changes and extensions concern Section 3. On top of text modifications, we introduced a new Table 2 that presents the signatures of DSL functions/operators (it is a shortened version of a table that was in the Appendix of the original submission, now Table 8) and added two new figures 4 and 5, which elucidate the process of program execution and program synthesis.

In addition to the changes required and requested by the Reviewers, we carefully proofread the manuscript, polishing the language, simplifying the text, and introducing minor reorganizations (in particular, moving some statements to the experimental sections).

For your convenience, **this PDF file contains both the answers to Reviewers and, attached at the end, the result of ‘Latex diff’ between the original submission and the revised one, where the added fragments are shown in blue and the deleted ones in red.**

We hope that the introduced extensions and amendments will meet your expectations.

With kind regards,  
Jakub Bednarek and Krzysztof Krawiec

## Review #1

Recommendation: Minor revision

Detail Comments

The paper presents a neurosymbolic system designed to tackle and solve tasks from the ARC dataset. The system architecture is modular, combining symbolic and subsymbolic components, including a DSL, to generate synthetic tasks and iteratively train a model to solve a subset of these generated tasks alongside the original tasks. The paper presentation and organization are good. The language is correct with only minor revisions required. The work also explains why the research should be placed in the scope of neurosymbolic systems.

*Thank you for your overall positive opinion on our work.*

Some points to raise that require the attention of the authors and might need addressing:

**[R1]** Section 3.2 describes the solver's **distributional generation** of programs as solutions to given tasks and mentions capturing the relevant semantics needed in addition to the syntax for solving tasks. **I am unsure to what extent language-building with grammars can be circumvented and how robust probabilistic semantics might be in replacing them.** It seems like there remains a **missing link** here between syntax and semantics that is not thoroughly covered by the methodology employed.

**Concerning “circumventing the language-building with grammars”:**

*For the avoidance of doubt, let us start by stating that our DSL has a fixed, formal grammar – it is just that we didn't present it in the paper. for the reasons outlined below. The grammar of the DSL remains fixed during training, and TransCoder complies with it by design, i.e. it cannot generate a syntactically incorrect program, by construction. What changes in training is only the engine/algorithm that synthesizes programs.*

*We did not present the formal grammar of our DSL in the original submission, and we do not do it in the revised version. The reason is that it is type-parametric (due to the presence of generic types) and thus not context-free, so it looks quite convoluted when written down formally. We find it more natural and convenient for the reader to express it implicitly, using the types presented in Table 1 and operations shown in the newly introduced Table 2, which is a shortened version of Table 8 from the appendix. We also provided an extended description of the DSL and the implicit grammar in the revised version.*

**Concerning “how robust probabilistic semantics might be in replacing them [grammars]”:**

*Programs expressed in our DSL have deterministic, well-defined semantics: there are no random effects in program execution, so a program applied to a given input always produces the same output. We use the term ‘semantics’ quite informally in the paper, as formalizing the semantics of our DSL would be even more complex and verbose than for the grammar.*

*What is non-deterministic in TransCoder is [part of] the program synthesis process, due to the presence of the variational layer in the model. As a result, TransCoder may come up with different proposed programs when queried multiple times on the same task, but those programs will still belong to the adopted DSL. This mechanism has been intentionally introduced to improve exploration during training, and can be switched off for test-set querying, should that be necessary, making the synthesis process deterministic (we also remark on in the text). However, this does not affect the semantics of program execution – that remains deterministic.*

*To address the doubts that may arise around these aspects, in the revised version of our submission, we extended the text at places (in the Sec. 3.2 indicated above and elsewhere) to convey the above characteristics in a more lucid fashion. We hope that these changes provide the ‘**missing link**’ between syntax and semantics of our DSL. We also added two new diagrams, in Figures 4 and 5, which should help convey the process of program generation and execution.*

[R2] Section 3.4 details the DSL created for the purpose of the experiments and the tasks, and the language seems like an interesting **bridge between lower-order types and higher-order functions**. But since the **DSL must deal with different kinds of operations (e.g., general arithmetic versus domain-specific filtering or rotation)**, it is difficult to see the motivation behind creating a DSL instead of using existing methods like **Inductive Logic Programming or existing languages like Prolog to do the job**. The motivation for implementing the language from a practical (i.e., this is a language that will have to scale if applied to other problems) and effective perspective (i.e., newly created and for the purpose of this task versus more robust pre-existing languages that are proven to handle logical problem framing and solving) does not seem sufficiently strong and persuasive.

*Thank you for this insightful question. There are a few arguments in favor of using a bespoke DSL rather than relying on generic languages and approaches like ILP or programming in logic. We outline them here, and decided to extend the manuscript with similar arguments at the end of Section 3.4 (admittedly, our original submission was somewhat bit scant in this respect).*

- 1. Our dedicated DSL is equipped with an adequate type system from the very start. There are specialized types for coordinates, regions of connected pixels, and generics (like Pair). This domain-specific knowledge is essential for efficient training, especially in the initial phases of the process, where the model struggles to synthesize programs capable of ‘doing anything interesting’.*
- 2. A fair share of ARC puzzles are ‘operational’ in nature, in the sense that they require the input panel (raster image) to be somehow transformed into the answer panel. It is thus natural to express such transformations as executable sequences (or other control structures) of steps/instructions that change some initial state (often given by the input panel) into some kind of dependent state (e.g. the output panel). Examples include moving objects, connecting points, mirroring fragments of the input image, counting objects in the input image and ‘expressing’ the obtained number in the output raster, and more. Expressing operations/transformations of this kind in ILP or Prolog, which are declarative rather than imperative, while hypothetically possible, would be much more cumbersome.*
- 3. Given the above arguments, we don’t feel entirely convinced that “the pre-existing languages are more robust”. We definitely agree that they are more general and expressive than our DSL, but there’s a price to pay for that, as argued above: starting from scratch would make the program synthesis task significantly harder.*

[R3] Section 3.6 presents the training of TransCoder. It is explained that the RL method is not useful at the start of training and kicks in later. **It would have been interesting to have a comparison between using RL versus using sole SL training which is more straightforward to implement and effective from the start. It would also help to convince of the criticality of having the RL method on top of the SL method.**

*Perhaps our wording was not precise enough in that paragraph. What we meant is that RL did not prove particularly effective at any stage of training, whether the initial one or the later*

one (with emphasis on the former). We have now reworded the beginning of Sec. 3.6 to make this clear.

*The gap between the effectiveness of SL and RL we observed in preliminary experimenting was so significant that we gave up RL altogether in this particular study. At the moment we find it very unlikely for any variant of RL to be useful for training our TransCoder – given the sparsity of positive rewards (no rewards for programs that do not solve the problem) and the complicated structure of the search space (minor differences between programs corresponding to fundamental changes in their behavior/semantics).*

*We also think that the generative mode proposed in this paper provides a much more elegant and effective avenue towards effective training of TransCoder (and similar architectures).*

**[R4]** Section 4 mentions related works and addresses LLM technologies, stating that their advancement has enabled better performances at solving ARC tasks than any DSL. This claim raises questions: **why haven't LLMs been included in the experiments and compared as benchmarks to TransCoder?** Why wasn't LLM technology considered as part of the design of the TransCoder modules? They would have possibly made for a less tedious implementation and more performant solution than a DSL according to the stated literature.

*There are multiple reasons why LLMs are not part of this study (even though we discussed them at the end of Sec. 4 in the original submission):*

- 1. LLMs do not guarantee the synthesized programs to be syntactically correct. TransCoder, in contrast, assures it by explicitly engaging DSL grammar in the process of program synthesis, when the DRNN traverses the AST of the program being generated. Having this guarantee makes the approach more elegant and computationally efficient, as the programs don't need to be additionally checked for syntactic consistency.*
- 2. LLMs are known to be flawed in many ways: confabulating, unpredictable, hungry for computing resources, etc.*
- 3. ARC does not involve natural language. While this does not preclude LLMs from being used here, that requires additional 'tinkering'. See, for instance, <https://arcprize.org/blog/oai-o3-pub-breakthrough> for the attempts of solving ARC problems with LLMs.*
- 4. There are arguments for claiming that natural language is not best fitted to solve ARC problems by, among others, being too informal and 'coarse' and struggling to capture certain forms of regularities. The operations contained in our DSL are much more principled in this sense.*
- 5. We (i.e., the scientific community) do not really know how LLMs work. Embedding them into the kind of architecture like TransCoder would not explain anything: we would have a black box generating candidate solutions. No explanations, no understanding of the 'inner workings', no actionable insights. In contrast, TransCoder is modular, with each module having a well-defined role and place in the overall*

*architecture, opening the door to proper understanding of its internal operation (not attempted in this paper for brevity, but certainly possible).*

6. *Last but not least, in this project, we strive at achieving insightful scientific results, rather than exercising pure engineering. With this in mind, we are studying the capacities of learning agents, rather than picking the tools/algorithms/methods that are expected to 'work better'.*

**[R5]** In Section 5, there is **mention of a normalization step without real justification of the importance of this step beyond grouping code into other code logic**. The authors mention reducing lines of code but in the examples given it seems more like code substitution and a tedious and unnecessary process to include.

*We have added in Sec. 5 a description that motivates the need to have a normalization mechanism integrated with the process of generating new training examples and presenting them during the training cycle. In particular, we have described the situation in which the generator learns redundant expressions and prioritizes their use during inference. Detecting redundant operations during program generation is difficult (it requires full knowledge of the subroutines, as in the case of Union type, where both subroutines must be the same), therefore we decided to carefully select the training data to minimize the probability of such situations.*

**[R6]** Figure 5 is hard to read and may be more exploitable if separated into 2 figures (one for SolveRate and one for SynthRate).

*We have separated SolveRate and SynthRate into two distinct graphs (now in Fig. 7) while maintaining the same x-axis scale for efficient analysis of events occurring during experiments.*

**[R7]** Table 5 is hard to read and I am not sure how to understand it. The explanation seems to suggest there is gradual learning (including of historical tasks), but the table presentation makes it very hard to render these kinds of conclusions.

*We have extended the caption of this table (Table 6 in the revised manuscript) with a broader explanation of the results achieved by the model snapshot from a given training moment.*

## Review #2

Recommendation: Major revision

Detail Comments Undoubtedly, this is a very interesting paper. Essentially, along the lines of Dreamcoder, the paper proposes something called coder. The idea is to use program synthesis for abstract task generation and solving. Like Dreamcoder and a host of other papers, they introduce a domain-specific and discuss how it can provide the constructs to

train a program to solve these reasoning tasks. They tackle the ARC reasoning tasks, which are challenging and worthy of consideration. Overall, in terms of the project's goals and ambition, I have issues at all.

*Thank you for your overall positive opinion on our work.*

**[R8]** What I do somewhat struggle with is that, unlike those other papers, this article is specifically submitted to the Neurosymbolic Journal. In this case, I would expect a bit more emphasis on the formal machinery behind all of these constructions. The way the paper is introduced is somewhat high-level, presenting some machine learning constructs and terminology without much detail. In fact, the paper is largely textual. Given the venue, I would expect a greater emphasis on semantics, syntax, and the assumptions behind some of the modeling of the distributions and the correctness of the whole thing.

*Thank you for this insight. This remark seems to strongly resonate with the sentiments of Reviewer #1 (especially in his/her first two remarks, [R1] and [R2]). Please refer to our answers to those remarks above. Overall, we have significantly extended our submission to cover the approach, and in particular the language, in greater detail (Sec. 3 and new figures 4 and 5).*

**[R9]** I understand that as a proof of concept, the pipeline is coming together nicely, but as a formal object, **there's much to be desired in the writing**.

*In addition to addressing the specific remarks by the Reviewers (most of them concerning Section 3, presenting the approach), we revised the entire paper carefully, and introduced multiple improvements in presentation and language. Hopefully those will meet your expectations.*

**[R10]** I recommend that the authors stay with this piece of work but try to provide some kind of soundness, completeness, or some sort of formal structure to how the various pieces fit together. I understand that this is challenging, but that's what would make it worthwhile for this journal.

*Admittedly, our proposed architecture is quite complex, and presenting it clearly is a challenge – especially without overwhelming the reader with too many technical details. We are quite confident that the revised version reads better and conveys our ideas more clearly.*