We thank the reviewers for their insightful remarks and suggestions. We have addressed them in the new version of the paper. We now reply to the comments and questions raised by the reviewers, starting with common questions.

[R2] [R4] Ablation studies. We conducted ablation studies and found that pretraining is critical: models trained from scratch fail to generate correct code. The DAE task is critical to initialize the decoding process (otherwise the model never understands it has to decode, and the back-translated sentences are too noisy to give a learning signal), but we found that it is possible to stop it after a few thousand iterations without impacting the performance. It is likely that using more powerful models like T5 would achieve a better performance. However, the back-translation step requires to translate functions on-the-fly at each iteration and using larger models significantly slows down the training (much more than on classification tasks). For instance, using a 24-layer decoder for generation would be too slow, but mixing a large encoder with a small decoder may be an option. In the context of natural languages, BART is trained with extra tasks such as span prediction or sentence permutation. These tasks could easily be adapted to programming languages and would be a promising direction for future work.

[R2] [R4] Supervision. For the pairs of programming languages we consider, we did not find any parallel datasets large enough to be used for training. Consequently, we are not able to make comparisons with supervised approaches. However, we agree this could be very valuable and it could be done in future work on language pairs where parallel datasets exist (e.g. CoffeScript $\leftrightarrow$ JavaScript). Moreover, a large parallel dataset could be useful to improve the pretraining: As shown in Lample and Conneau (2019), pretraining with both masked-language modeling and translation language modeling (TLM) objectives leads to a better performances in natural languages as the TLM objective provides high quality cross-lingual embeddings.

[R2] [R3] Function length v.s. performance. For C++ $\rightarrow$ Python, the length of functions in the test set varies from 16 to 430 tokens. We observe that the performance decreases when the length increases. For beam 10, the performance on functions with less than 45 tokens is of 72% accuracy, 30% accuracy for functions between 100 and 120 tokens, and 10% accuracy for functions with more than 210 tokens. Thank you for suggesting this study. It is very interesting and we will report the full table in the updated version of the paper.

[R1] Notations (Beam / Beam top-1 / CA@N) What we refer to as "Beam N - Top 1" is indeed what people refer to as "Beam N" in machine translation. We agree with the reviewer that this is confusing and we will update the paper with standard notations and the recommended "CA@N" instead of "Beam N", to highlight that we are using a different metric and to be consistent with the machine translation terminology.

[R1] Negative Broader Impact Discussion. We believe that the main negative consequence of developments in programming languages translation would be a reduced employability for experts in archaic programming languages. It is true that relying on ML-generated code could make IT systems more fragile, especially if the output of the ML system is not human-readable. Besides, programmers might have too much confidence in the translator and fail to spot errors they would have not made without the ML system. We will give it more thoughts in the Broader Impact section.

[R3] Robustness to method and variable names We manually tried to fool the model by providing input functions with inconsistent names, for instance by renaming a function called "factorial" to "fibonacci" (or something totally unrelated with the content of the function), or renaming a string variable to "number". We observed that the model is robust to these modifications, and that this do not impact the correctness of translations. Instead, TransCoder properly adapts the output to be more consistent with input names and types (c.f. Figure 8 in the appendix).

[R3] Challenges in programming languages translation. We agree that the paper would benefit from more elaborate discussions about issues and challenges in programming languages translation. Reacting appropriately when the source language uses a library with no equivalent in the target language is notably difficult. We did not observe this issue at test time because functions from GeeksForGeeks typically do not rely on external libraries (e.g. NumPy or SciPy), but this is indeed a current limitation of the model.

[R3] Language pair difficulty. TransCoder performs well on language pairs that share many keywords used for similar purposes (anchor points). Having a similar syntax helps, but TransCoder is also able to translate python-specific syntax to C++ or Java. The performance is lower when translating from Python. An explanation comes from the type inference, an additional difficult task that the model does not have in the other directions. As there are many common keywords between C++ and OCaml, and between Python2 and Python3, we expect that TransCoder would also perform well in these directions. The confusions between Python2 and Python3 because of the languages similarities should be mitigated by the use of the language token during decoding. It would be interesting to check whether the syntactic differences between C++ and a functional language would not be too much of a barrier.

[R4] Effect of higher beam sizes. We generated translations with a beam of size 50, but did not observe a very large difference compared to Beam 25. Improvements remain the most important between Beam 1 and Beam 5.