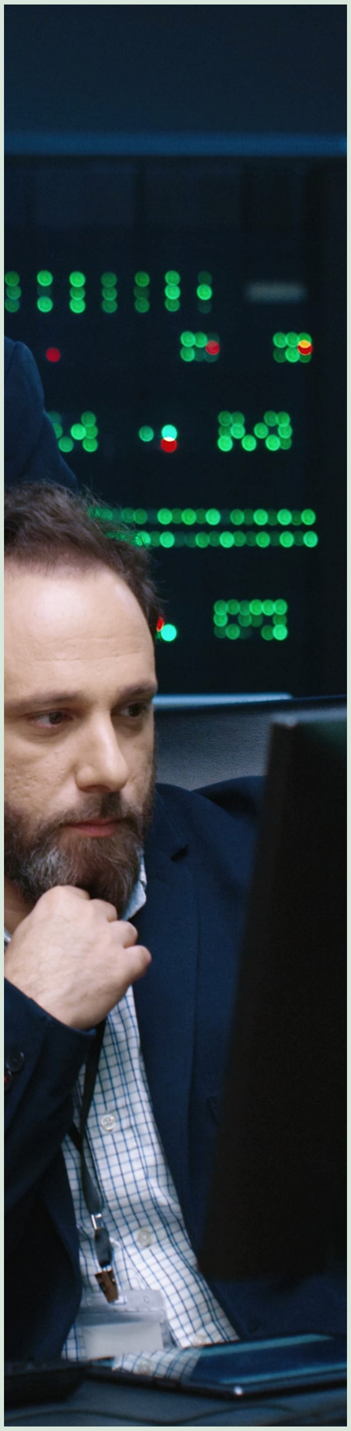
WHITE PAPER



DEVELOPERS OF THE FUTURE - ML BASED CODE GENERATORS

Abstract

The space of building software using specialised programming languages is ripe for disruption in the context of digital transformation. This paper will introduce the landscape and the advancement of emerging machine-learning models to direct translations between natural (NL) & programming language (PL) constructs with increasing confidence levels. The paper explores the trend at the vanguard of running bug-free translations on addressing complex circumstances while staying grounded on the steps the industry needs to take to keep on top of this transformational journey.

Infosys®
Navigate your next

Rising demand for multi-skilled developers

Organisations are adopting different programming languages to solve complex problems in their landscape. Of the ~600 programming languages in use, 240+ have emerged in the last eight years resulting in clients maintaining a hybrid technology ecosystem. In that world, it is getting difficult to find skilled talents both in the past and present for system maintenance & to run legacy modernisation programs. This ability continues to wane over time with a drop-in supply of skilled workforce in legacy technologies such as mainframes, and those who are experienced mainframe developers are unlikely to carry equal competence in some of the more recent paradigms.

So, the question is – How can one address this gap? Or a better question would be – How is this different from times before whenever technology has taken a quantum leap? The short answer is that technology assimilation has been organic thus far; however, now that has rocketed into the exponential category.

Current ways of working - Traditional approaches of Coding tasks automation

Typical Coding tasks are, Code generation from English description (NL-PL), Code translation between programming languages (PL-PL), Code documentation (PL-NL) and Code validation (generating unit tests) (PL-PL).

Traditional approaches of automation of PL-PL tasks relied heavily on their grammar. Since handling all possibilities in code via grammar is tedious, the conventional techniques managed most frequently occurring scenarios. Hence, limited coverage & success. Proprietary software does exist that claims to convert COBOL to C#/Java; however, the accuracy is likely less than 60%, implying months of manual labour. The demand was for alternate strategies that could scale quickly to handle more sophisticated scenarios.

Classic automation techniques of (NL) text-related tasks involve parsing the unstructured text as structured text with a poor degree of convergence as the unstructured text is amorphous in many ways. Hence, these methods also had lesser coverage, consequently lesser success. Accordingly, the demand matured for computing systems that could comprehend text as humans would.

Considering the future consisting of a complex hybrid technology landscape, how can organisations akin to Infosys build a community of future developers that could manage any technology combination in quick time with sufficiently high accuracy, sustainably?

Developers of the future – Adopting Pair programming with Code models

Developers of the past have had to code by referring manuals/books for syntax. Since then, developers use IDEs with IntelliSense, Transcompiler for the supported technologies, and google (code) search to find relevant code samples. Onus on appropriate usage of code samples from external websites (in Github/Stack Overflow), based on their licenses, has been with developers. From here on, we see the ML based NLP solutions driving this trend to this next level.

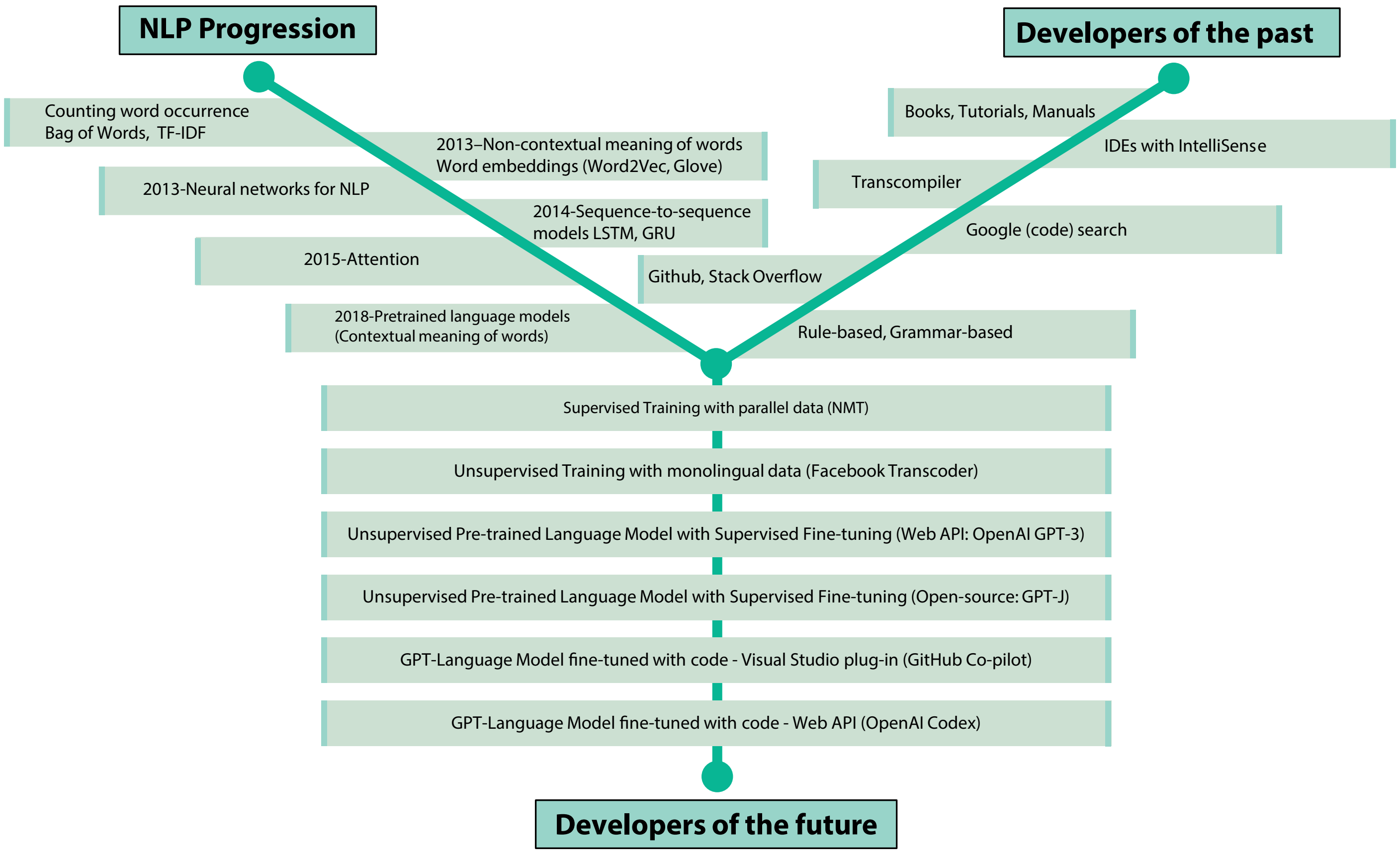
Machine learning-based NLP solutions do deem an alternative to rules/grammar based traditional NLP approaches, as it is easier to collect datasets than to write rules in augmenting coding tasks.

Such NLP systems have made colossal progress in comprehending text, from looking for the presence of words to classify documents to counting the occurrence of words (Bag-of-words, TF-IDF), progressing further towards understanding the non-contextual meaning of the words (Word2Vec [1] , GloVe [2]) and culminating in attaining the contextual meaning of words in the given sentence to excel at NLU tasks (BERT [3]) and NLG tasks (GPT-2 [5]). These state-of-the-art models mimic human-level accuracy in comprehending code descriptions written in English and generating code documentation.

Success in ML-based text processing led to progress in ML-based code processing. Traditional ML-based PL to PL tasks using Neural machine translation (NMT) involved (supervised) training models from scratch using task-specific labelled data, thus demanding a large quantum of training data, effort, and cost. However, supervised finetuning of un-supervised pre-trained models (EleutherAI's GPT-J, OpenAI's GPT-3) [4] have evolved to a point demanding lesser training data, effort, and cost, making it easier to adopt for all coding tasks. Employment of (supervised) few-shot learning on (unsupervised) pre-trained model (like, GPT-J, GPT-3) work well with very little training data and with no training effort and cost, suitable for tasks learnt well during pre-training. Even though models like GPT-2 [5] , GPT-J, GPT-3 [6] suit coding assignments best, they were built for the text-in-text-out category of tasks and not exclusively for coding tasks. OpenAI's Codex, a GPT language model, finetuned with code from GitHub, was constructed solely for coding tasks. Github CoPilot, powered by a distinct production version of Codex, is an extension for code editors. GPT-3 [6], Codex [8] and CoPilot, available as web-APIs, transfer the code to a 3rd party server for processing. While supervised machine translation models (NMT) for programming languages demands parallel data (Source PL statements and its equivalent Target PL statements) for training, Facebook's TransCoder [7] is an unsupervised machine translation model built with the bulk volume of monolingual code of source PL and monolingual code of target PL.

Considering the afore stated trends in ML-based code generation, our recommendation towards building developers of the future includes

- Build a team to specialise in enhancing pre-trained ML code generators to gain efficiency in each domain or use-case (task),
- Evangelise code generators through training and practice by adopting internally as their pair programmers in building their projects to champion and measure efficiency gain,
- Start strategising archival of code & related assets to enhance relevant models in the future.


NLP Progression
Counting word occurrence Bag of Words, TF-IDF
2013–Non-contextual meaning of words Word embeddings (Word2Vec, Glove)
2013-Neural networks for NLP
2014-Sequence-to-sequence models LSTM, GRU
2015-Attention
2018-Pretrained language models (Contextual meaning of words)
Developers of the past
Books, Tutorials, Manuals
IDEs with IntelliSense
Transcompiler
Google (code) search
Github, Stack Overflow
Rule-based, Grammar-based
Supervised Training with parallel data (NMT)
Unsupervised Training with monolingual data (Facebook Transcoder)
Unsupervised Pre-trained Language Model with Supervised Fine-tuning (Web API: OpenAI GPT-3)
Unsupervised Pre-trained Language Model with Supervised Fine-tuning (Open-source: GPT-J)
GPT-Language Model fine-tuned with code - Visual Studio plug-in (GitHub Co-pilot)
GPT-Language Model fine-tuned with code - Web API (OpenAI Codex)
Developers of the future

Conclusion

Our confidence to recommend the solution stems from history. Clients, over time, accepted cloud VMs (Cloud is virtually our system, but somebody's hardware) to store their data; Clients then progressed to use cognitive APIs (web-APIs) for processing their image files and audio files in 3rd party server.

Soon, the industry will accept code generators regardless of whether they are available as Web-API (like Codex) or as they are built with open-source training data (like GitHub Code) or as enhanced open-source code generators (like GPT-J) and we as industry veterans must plan to capitalise the trend.

References

1 Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013).

2 Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 1532–1543.

3 J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805, 2018.

4 A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, Improving Language Understanding by Generative Pre-Training, OpenAI, Tech. Rep., 2018. [Online]. Available: https://openai.com/blog/language-unsupervised/

5 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

6 Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. (2020).

7 Roziere, Baptiste and Lachaux, Marie-Anne and Chanussot, Lowik and Lample, Guillaume. 2020. Unsupervised translation of programming languages. Advances in Neural Information Processing Systems.

8 M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P.Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. HerbertVoss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, Evaluating Large Language Models Trained on Code, arXiv:2107.03374 [cs], Jul. 2021, arXiv: 2107.03374.

About the Authors

Kamalkumar Rathinasamy
Principal Technology Architect

Shreshta Shyamsundar
Senior Principal Technology Architect

Varsha Jain
Data Scientist

Balaji Jayaram
Data Scientist

Infosys®
Navigate your next

For more information, contact askus@infosys.com