A Natural Language Interface for Building Business Automation Decision Rules

Michael Desmond, Evelyn Duesterwald, Vatche Isahagian, and Vinod Muthusamy

IBM Research AI, USA
mdesmond@us.ibm.com, duester@us.ibm.com, vatchei@ibm.com, vmuthus@us.ibm.com

Introduction

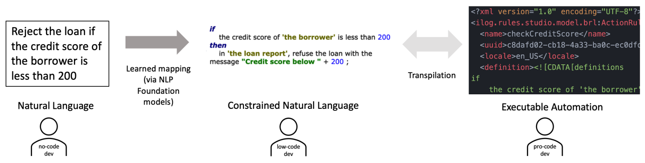
Business automation is a multi-billion dollar industry (Gartner 2021). Much of the automation software has been developed using traditional automation technologies. Consider for example operational decision management (ODM) (Boyer and Mili 2011) which enables users to define rules that automate and govern business decisions for domains such as banking (authorizing a loan) and insurance (approve an insurance claim). These applications come with a custom automation language, such as DMN (Biard et al. 2015), that requires users to have adequate programming and infrastructure skills to develop, debug and deploy. Business users are experts in their fields, but often lack the required skills to code and debug using these languages and tools designed for proficient developers.

Recently, there have been advances in synthesizing code from natural language (NL) descriptions using large language models (LLMs) (Poesia et al. 2022). Tools such as Copilot (Chen et al. 2021) can be integrated into editors to assist programmers by synthesizing code from NL comments. While these tools can provide productivity boosts for developers they still require familiarity with the target language in order to validate and fine-tune the synthesized code. In the case of business automation, users would still need to be able to review low level automation code, a skill, many business users don't posses.

We propose an NL paradigm and platform for the construction of business automation rules that incorporate a constrained natural language (CNL) – a domain-specific highly-consumable language to validate and review synthesized code. Our approach utilizes LLMs to translate business rules described in NL into CNL for human review which is then further transpiled into the business automation code of the rule engine. To address challenges in the translation from NL to CNL, we utilize several techniques such as constrained decoding (Shin et al. 2021), fine-tuning and prompting (Liu et al. 2021).

Design and architecture

Our paradigm for NL based automation is illustrated in Figure 1. We connect an NL based no-code front end to a target


Reject the loan if the credit score of the borrower is less than 200
Learned mapping (via NLP Foundation models)
Natural Language
no-code dev
Constrained Natural Language
low-code dev
Transpilation
Executable Automation
pro-code dev

```
if
   the credit score of 'the borrower' is less than 200
then
   in 'the loan report', refuse the loan with the
message "Credit score below " + 200 ;
```

```
<?xml version="1.0" encoding="UTF-8"?
<ilog.rules.studio.model.brl:ActionRul
  <name>checkCreditScore</name>
  <uuid>c8dafd02-cb18-4a33-ba0c-ec0dfc
  <locale>en_US</locale>
  <definition><![CDATA[definitions
if
    the credit score of 'the borrower'
```

Figure 1: Illustrative NL, CNL and automation code example of a business automation rule.

automation language backend via an intermediate CNL. The CNL bridges the semantic gap between the citizen developer's NL input and the low-level automation code. CNLs are non-ambiguous domain specific languages (DSLs), but CNL domain keywords and abstractions bear more resemblance to NL and hence tend to be easy to read and understand. However, the CNL is conceptually not any easier than learning other DSLs and is generally not easy to write from scratch. Figure 1 provides an example illustrating the NL, CNL, and automation code representation of the same business rule example.

The intermediate CNL provides several advantages. First, it enables the rapid construction of learned NL to CNL mappings based on LLMs. Second, as an easy-to-read language, it allows the citizen developer to review and validate whether they achieved what they had in mind. Third, as a DSL, it facilitates the construction of source-to-source transformation (transpilation) to the underlying automation code using traditional compiler technology (Kulkarni et. al 2015).

Our approach provides a combined no-code/low-code development experience where business users use NL to build automations that can be reviewed at the CNL level before being transpiled to the underlying automation code. Another advantage targets proficient developers: they can use the NL front end to quickly bootstrap an initial automation that can be edited and refined at the automation code level.

In order to construct the NL front end, we need the ability to quickly learn a translation from the NL input to the corresponding CNL representation. We use LLMs to construct these learned mappings with minimal training data (Bommasani et al. 2021). As shown in Figure 2, our architecture supports two options. In one case, a pre-trained model, such as GPT-Neo, is used along with a prompt generation component that performs a neighborhood query at inference time to select the training data as part of the prompt engineering

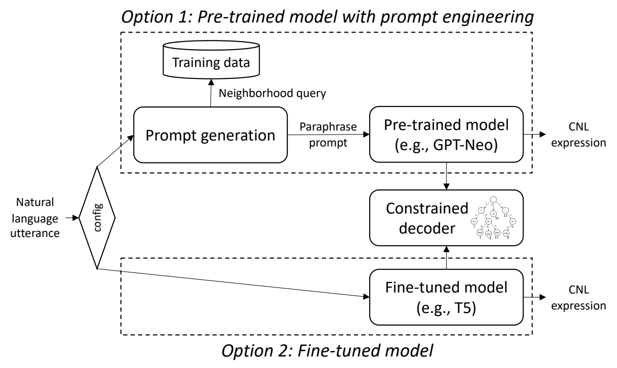

Option 1: Pre-trained model with prompt engineering
Training data
Neighborhood query
Prompt generation
Paraphrase prompt
Pre-trained model (e.g., GPT-Neo)
CNL expression
Natural language utterance
config
Constrained decoder
Fine-tuned model (e.g., T5)
CNL expression
Option 2: Fine-tuned model

Figure 2: Using either fine-tuned LLM, or pre-trained ones with prompt engineering to map NL utterances to CNL expressions. Constrained decoding is optional.

for the model. In the other case, a language model (such as T5) is fine-tuned offline on the training set. In both cases, a constrained decoder can optionally be enabled to ensure that the output from the model conforms to the CNL grammar. The constrained decoder can either be implemented by deriving a prefix-tree (Shin et al. 2021) from the training data, or if available by incorporating a grammar of the CNL (Poesia et al. 2022). The training data is a corpus of (NL, CNL) pairs. Data augmentation approaches can be applied to this dataset, such as NL paraphrasing (Anaby-Tavor et al. 2020).

Our experiments show the promise of the proposed approach, and our demo allows users to experience the responsiveness and accuracy of the architectural options.

Evaluation

We evaluated the efficacy of NL to CNL translation using a variety of LLM configurations and the use of constrained decoding, fine-tuning vs. prompting, and varying availability of training data. We experimented with the miniloan[1] business rules dataset, which includes a set of CNL business rules with corresponding NL paraphrases. We compared the following LLM configurations:[2] T5 and GPT-Neo-2.7B. The T5 model was fine-tuned using training data, the GPT model was prompted at inference time with NL-CNL samples from the same training data. We implemented a prompt generation pipeline to generate a prompt based on the similarity of an input NL statement to those in the training data. We implemented constrained decoding using a prefix tree, constructed from tokenized examples of valid CNL statements.

The data set was split into 70% train, 24% test and 6% validation sets. The predictions from each of the pipelines were then evaluated against the reference CNL statements from the test set to produce results. The primary metric we report on is accuracy. We also evaluated with a reduced training set of only 100 samples and fine-tuned another version of T5 using the reduced dataset.

As shown in Table 1, fine-tuned LLM configurations performed better than prompted GPT configurations. T5 with constrained decoding performed best. This observation held

[1]https://github.com/DecisionsDev/odm-for-dev-getting-started

[2]Additional experiments are available at (Desmond et al. 2022).

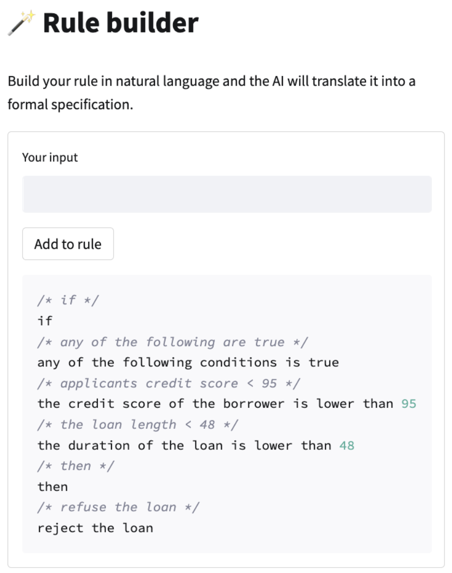
Dataset	miniloan-full			miniloan-100
Metric	INF	ACCURACY		
T5	.23	.98	.91	.86
T5/C.	**.29**	**.99**	**.98**	**.87**
GPT2.7B	4.9	.43	.26	.60
GPT2.7B/C.	5.0	.76	.63	.63

Table 1: Accuracy (exact match prediction) across language model configurations. Each LLM ran without and with constrained decoding (indicated with **/C.**). Miniloan-100 indicates the reduced training set was used. INF indicates inference time in seconds.

even when comparing fine-tuned LLM configurations with limited training data. Constrained decoding provided slight performance improvements over fine-tuned LLM configurations with full training data. Inference time became progressively slower with GPT taking 5 seconds per example.

Demo Overview

The demo models the use case from Figure 1, where a business user interacts with the system to construct a decision rule using only natural language. As shown in Figure 3 the user incrementally builds a decision rule in an iterative development style. The user controls the size of the input increments and receives immediate feedback as the CNL code synthesized by the LLM. The CNL feedback is easy to read enabling quick validation without the need to examine the low level decision rule format. To explore performance variations across different LLMs we made the demo configurable with custom backend LLM choices.



Build your rule in natural language and the AI will translate it into a formal specification.

```
/* if */
if
/* any of the following are true */
any of the following conditions is true
/* applicants credit score < 95 */
the credit score of the borrower is lower than 95
/* the loan length < 48 */
the duration of the loan is lower than 48
/* then */
then
/* refuse the loan */
reject the loan
```

Figure 3: Demo view providing a snapshot in the process of constructing a decision rule from natural language input.

Our iterative development approach enhances productivity by allowing the user to break down complex rules into custom sized chunks. The demo illustrates an important synergy between model performance and user productivity in that smaller user input increments can lead to both better model performance and improved human productivity.

References

Anaby-Tavor, A.; Carmeli, B.; Goldbraich, E.; Kantor, A.; Kour, G.; Shlomov, S.; Tepper, N.; and Zwerdling, N. 2020. Do not have enough data? Deep learning to the rescue! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 7383–7390.

Biard, T.; Mauff, A. L.; Bigand, M.; and Bourey, J.-P. 2015. Separation of decision modeling from business process modeling using new "Decision Model and Notation"(DMN) for automating operational decision-making. In *Working Conference on Virtual Enterprises*, 489–496. Springer.

Bommasani, R.; Hudson, D. A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M. S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Boyer, J.; and Mili, H. 2011. IBM Websphere ILOG JRules. In *Agile business rule development*, 215–242. Springer.

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

Desmond, M.; Duesterwald, E.; Isahagian, V.; and Muthusamy, V. 2022. A No-Code Low-Code Paradigm for Authoring Business Automations Using Natural Language. ArXiv:2207.10648 [cs].

Gartner. 2021. Gartner Forecasts Worldwide Hyperautomation-Enabling Software Market to Reach Nearly $600 Billion by 2022. https://www.gartner.com/en/newsroom/press-releases/2021-04-28-gartner-forecasts-worldwide-hyperautomation-enabling-software-market-to-reach-nearly-600-billion-by-2022.

Kulkarni et. al, R. 2015. Transpiler and it's Advantages. *International Journal of Computer Science and Information Technologies*, 6(2).

Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; and Neubig, G. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

Poesia, G.; Polozov, O.; Le, V.; Tiwari, A.; Soares, G.; Meek, C.; and Gulwani, S. 2022. Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*.

Shin, R.; Lin, C. H.; Thomson, S.; Chen, C.; Roy, S.; Platanios, E. A.; Pauls, A.; Klein, D.; Eisner, J.; and Van Durme, B. 2021. Constrained language models yield few-shot semantic parsers. *arXiv preprint arXiv:2104.08768*.