

Generating Infrastructure-as-Code from Natural Language: Evaluating Fine-Tuned GPT-3 Models for Cloud Infrastructure Provisioning

Stanford CS224N Custom Project

Yevgeni Chuvyrov
Department of Computer Science
Stanford University
chuvyrov@stanford.edu

Abstract

Infrastructure-as-Code (IaC) is a key enabler of "digital transformation," where enterprises are re-imagining their business models by putting technology and cloud computing first. IaC is code that describes infrastructure to be created, such as, for example, an Azure Data Science Virtual Machine with Nvidia GPU that we used in class, or a massive cloud-based data warehousing solution required by many enterprises. Any cloud infrastructure can and should be defined as an IaC because it is a reliable, reusable and scalable way to create and manage cloud resources. However, creating IaC artifacts requires not only in-depth knowledge of infrastructure concepts, but also a significant software engineering experience. To make cloud computing more accessible, I introduce a transformer-based model to generate IaC from natural language, which allows both technical and nontechnical users to dynamically generate IaC artifacts, enabling them to request and receive cloud resources using conversational interfaces such as chat bots, SMS, etc. My GPT-3-based models are fine-tuned on a set of 5,182 Infrastructure-as-Code templates specific to Microsoft Azure. BLEU scores on the test set indicate that this could be a viable way to deploy cloud infrastructure in the near future.

1 Key Information to include

- TA mentor: Gaurab Banerjee
- External collaborators (if no, indicate "No"): No
- External mentor (if no, indicate "No"): No
- Sharing project (if no, indicate "No"): No

2 Introduction

Today, virtually all enterprises are re-inventing themselves around the concept of "digital transformation," which is essentially a re-imagining of their business models by putting technology first. The origin of digital transformation is tied directly to the rise of cloud computing, where all technological resources (hardware and software) are managed by cloud providers, who leverage economies of scale to offer cutting edge computing at accessible prices. However, provisioning and using cloud resources at enterprise scale today is difficult, especially for nontechnical users.

Most frequently, highly specialized skills are required to create what is known as Infrastructure-as-Code (IaC) artifacts before any cloud resource can be provisioned in an enterprise setting. By definition, IaC is code that describes individual pieces of cloud infrastructure and how they relate to each other. As code, IaC can be versioned, reused and changed on demand to upgrade or rollback

infrastructure, from simple Virtual Machines to massive multi-node data warehousing clusters. While GUIs, such as web-based cloud portals, are being constantly upgraded to reflect the wealth of new technologies and features constantly introduced in the cloud, it is close to impossible to deploy sophisticated cloud infrastructure using only web interfaces today. And even simple infrastructure, such as Virtual Machines with GPUs attached requires multi-page instructions and screen shots to be deployed using the web, as we have learned in class trying to use Azure for our NLP work.

In this project, I aim to create an NLP system that generates code (IaC artifacts) using natural language queries. For example, a query of "I need a Virtual Machine with GPU" should produce code that is close to being deployable into the cloud and capable of creating a Virtual Machine. I use publicly available IaC templates with corresponding metadata to create a pre-processing pipeline to generate data necessary for training. I then use this data to fine-tune the state-of-the-art GPT-3 transformer model using its newly released API. Finally, BLEU scores and human evaluation are used to determine the performance of this fine-tuned model.

By following these approaches, I observe that my models are capable of generating IaC from natural queries. According to my evaluation metrics, my models appear to be successfully simplifying the effort required to get users cloud resources they need, potentially improving productivity and accelerating the digital transformation journey. While the focus of this project is specifically on Microsoft Azure, the concept can be extended to any number of cloud providers, assuming there's enough training data available.

3 Related Work

3.1 Code Generation

There have been efforts to solve the problem of code generation or transpilation from one programming language to another for as long as there have been attempts to solve the problem of Machine Translation. And, similar to the way neural methods have dramatically improved Machine Translation accuracy, they have also renewed interest and significantly improved our ability to generate code. In [1], a Transformer with 6 layers, 8 attention heads was used to create a model to translate one programming language to another (i.e., Python to C++). In [2], the authors used open-source GPT-2 model fine-tuned on Python code to complete a function based on its description, and in [3] they used Transformer architecture to generate code from pseudocode provided. In [4], given a large team and a presumably a very large budget, the authors have pretrained a multi-layer generative transformer model for code (GPT-C), which is a variant of the GPT-2 trained from scratch on a large unsupervised multilingual source code dataset. To do that, it only took them 5 Lambda V100 boxes, each having sixteen V100 GPUs with 32 GB HBM2 memory, eight 100 GB InfiniBand, and one 100 GB Ethernet connection, managed with Kubernetes, or an approximately a \$1,000,000 budget.

I am not aware of efforts specifically related to infrastructure-as-code generation to date, with most research focused on generating Python or Java-like language from pseudocode or comments. However, these code generation efforts are similar to that of generating an IaC (see Appendix 1 for an example of IaC). The use of Transformers for these efforts has been proven superior to other methods such as LSTMs, since Transformer's ability to model long range dependencies appears to be better [7].

3.2 Code Generation Evaluation

All literature reviewed mentions BLEU score as a proper evaluation metric for code generation efforts. Several papers [2][6] suggest that there are limitations and improvements that can be made to the BLEU metric to better evaluate model performance. For instance, [6] finds that BLEU has problems capturing semantic features specific to code, and suggests several semantic modifications to the score.

4 Approach

4.1 Model

As Professor Manning mentioned in class and is also evident from recent research papers focused on code generation [2][3][4][5], today's Natural Language Processing revolves mostly around large pre-trained transformer models. OpenAI GPT-3 (175 billion parameters) is a closed-source successor to fully open source GPT-2 (1.5 billion parameters) pre-trained transformer model that can be considered state of the art by today's (February 2022) standards. As recently as December 2021, OpenAI opened a way to fine-tune GPT-3 text generation capabilities. For my project, I fine-tuned GPT-3 on the pre-processed Infrastructure-as-Code files specific to Microsoft Azure using GPT-3 ada engine. During the first phase of experiments, I evaluated using GPT-3 top of the line (and very expensive!) davinci engine, but as I expanded my dataset 7x, davinci engine became too cost-prohibitive. Once fine-tuning was complete, I then used GPT-3 text generation capabilities to create IaC files for data in my test set.

4.2 Data

For training, I used a GitHub repo with Azure QuickStart templates (available at <https://github.com/echuvyrov/azure-quickstart-templates>), which contains over 1,000 IaC artifacts for provisioning anything from a basic Virtual Machine to massive Kubernetes clusters with application workloads deployed onto them. Each IaC artifact is neatly packaged into a separate folder and includes an associated metadata file describing what that specific piece of infrastructure does. I was able to pre-process and reformat 700 of those templates for my training.

It's important to note that IaC files can get quite large when a lot of infrastructure elements are defined within them. To reduce code generation efforts to manageable chunks, I limited it to generating specific pieces of infrastructure only, without its supporting elements. For example, when the request is for a Virtual Machine, only the IaC element specifying the Virtual Machine is created, without the network card or a Public IP that may be required for that VM. Conceptually, this would be similar to generating a single function instead of the whole program based on natural language input.

The relaxation of the requirement to generate complete templates allowed me to significantly increase training data set size: from 700 templates available to 5,182 training samples and 576 samples (10%) held out for testing and evaluation of model performance. This was accomplished via multiple passes over the templates available looking for specific blocks of infrastructure to extract (i.e., a Virtual Machine element is likely present in the solution defining a multi-node Windows Active Directory clusters).

4.3 Evaluating Model Performance

As mentioned in [2][6], BLEU is a common metric used to evaluate code generation performance. I used the implementation of BLEU in NLTK package (`sentence_bleu`) to compare the expected output to the one produced for my test set, which was held out during training.

5 Experiments

5.1 Data

I have pre-processed and created a pipeline for packaging data into the format expected by GPT-3. The Azure QuickStart templates (<https://github.com/echuvyrov/azure-quickstart-templates>) define a metadata file for each piece of infrastructure in the QuickStart set. This example metadata file is shown below, with summary of what associated template does conveniently provided:

```
{
  "$schema": "https://aka.ms/azure-quickstart-templates-metadata-schema#",
  "type": "QuickStart",
  "itemDisplayName": "101-1vm-2nics-2subnets-1vnet",
```

```

    "description": "Creates a new VM with two NICs which connect to two different subnets.",
    "summary": "Creates a new VM with two NICs which connect to two different subnets s.",
    "githubUsername": "lmoxiel",
    "dateUpdated": "2021-07-07"
}

```

The "summary" property in the template above is used as a natural language prompt to create infrastructure. The associated IaC file is located within the same folder as the corresponding metadata; a portion relevant for code generation efforts (and related to the above metadata file) is pasted below.

```

{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2020-06-01",
  "name": "[variables('virtualMachineName')]",
  "location": "[parameters('location')]",
  "properties": {
    "osProfile": {
      "computerName": "[variables('virtualMachineName')]",
      "adminUsername": "[parameters('adminUsername')]",
      "adminPassword": "[parameters('adminPassword')]",
      "windowsConfiguration": {
        "provisionVMAgent": true
      }
    }
  }
}
[..]

```

5.2 Evaluation method

I have withheld 576 samples (10% of training data) from the training data set and used the fine-tuned GPT-3 model I created to make predictions on the templates in that set. I then used BLEU metric to compare the expected output to the actual generated code.

5.3 Experimental details

I have fine-tuned GPT-3 model with my data using GPT-3 ada engine with 5,182 IaC templates. Initially, using a much smaller data set of 700 templates, I have also fine-tuned OpenAI state-of-the-art davinci engine. Unfortunately, fine-tuning with davinci gets to be expensive fast, as it cost \$50 for a single round of fine-tuning on only 700 samples.

I then run predictions on my held out test data using GPT-3 parameters of temperature = 0 (to make model very deterministic), response length = 1024, with other parameters left at default. Fine-tuning ran over the default 4 epochs.

5.4 Results

A test sample with a prompt, expected output and the output of my fine-tuned model returned is shown below. Note that this particular sample has BLEU score of 96.6%, which is far above the average for all templates.

Prompt

Creates and optionally secures a Key Vault with logging linked to a storage account

Expected Output

```

{'type': 'microsoft.keyvault/vaults',
 'apiversion': '2019-09-01',
 'name': "[parameters('keyvaultname')]",
 'location': "[parameters('location')]",
 'dependson': ["[resourceid('microsoft.managedidentity/userassignedidentities',
 parameters('identityname'))]"], 'properties': {'sku': {'name': 'standard',

```

```
'family': 'a'}, 'tenantid': \"[reference(variables('identityid')).tenantid]\",
'accesspolicies': [{ 'tenantid': \"[reference(variables('identityid')).tenantid]\",
'objectId': \"[reference(variables('identityid')).principalid]\", 'permissions':
{'secrets': ['get']}]}, 'enablesoftdelete': true}}
```

Actual Output

```
{'type': 'microsoft.keyvault/vaults',
'apiversion': '2019-09-01',
'name': "[parameters('keyvaultname')]\",
'location': "[parameters('location')]",
'dependson': "[resourceid('microsoft.managedidentity/userassignedidentities',
parameters('identityname'))]", 'properties': {'sku': {'name': 'standard',
'family': 'a'}, 'tenantid': "[reference(variables('identityid')).tenantid]\",
'accesspolicies': [{ 'tenantid': "[reference(variables('identityid')).tenantid]\",
'objectId': "[reference(variables('identityid')).principalid]\", 'permissions':
{'secrets': ['get']}]}, 'enablesoftdelete': true}}
```

A set of 576 such samples was used to calculate the BLEU score statistics for the test data set. An automated pipeline to extract the metadata and the associated template, as well as return GPT-3 output for a given prompt was created. Then, a set of BLEU statistics was created for the test set. These statistics are presented below.

Table 1: IaC Code Generation Results

IaC templates in training set	5,182
IaC templates in the test set	576
Test set: Highest BLEU score	96.76%
Test set: Lowest BLEU score	1.377e-231%
Test set: Average BLEU score	13.04%

While the average BLEU score of 13.04% does not indicate exceptional performance of our fine-tuned model, it's a good starting point to analyze where the model does extremely well or extremely poor, as well as plan for how to improve this metric.

6 Analysis

Experimental results show a wide range of BLEU scores for the test data: from virtually 0 all the way up to 96.76%. Closer examination reveals that model does well on generating relatively short, self-contained pieces of infrastructure with limited number of variations. For example, Azure Key Vault, which allows for secure storage of secrets in the cloud, or Azure HDInsight cluster, which is a managed version of the big data platform Hadoop, do very well using our fine-tuned model. Both of these IaC pieces are fairly self-contained with a small number of properties. For this specific category of infrastructure, our model is able to learn proper translation mechanics between the natural language query and its IoC representation.

On the other end of the spectrum, lowest BLEU scores are usually associated with the relatively obscure pieces of infrastructure, such as creating alerts on Application Insights or a PostgreSQL datastore on an Azure Machine Learning workspace. This is likely an indication that this is the first or close to first time the model is seeing this specific bit of infrastructure requested and doesn't know exactly what to construct. To address limitations in this area, we will have to find more data for the model to work with.

In the average scores category, there are common infrastructure elements with a wide range of possible parameters. For this category, while the model seems to be able to generate basic IaC representation of the natural language, it's struggling to address a wide variety of parameters requested by the natural language input.

7 Conclusion

As my experiments with GPT-3 show, it is possible to generate Infrastructure-as-Code artifacts today using natural language. Although these generated artifacts may not be in a fully deployable state yet, they represent the first step towards cloud resource generation using natural language. With certain tweaks to the model in the form of additional data, better manual labeling of infrastructure, as well as more powerful (but much more expensive) GPT-3 engine (davinci), we can start seeing improvements almost immediately.

While these GPT-3-based results are somewhat encouraging, GPT-3 is a closed source system with only so many knobs to tweak. It could be worth evaluating how a fully open source GPT-2 model (or a similar open source model) fine-tuned on the same data set compares to the performance of GPT-3.

Finally, [5] recommends using functional testing for evaluating code generation performance in addition to BLEU scores. For IaC, this functional testing would take the form of taking a template produced and trying that deployment against Microsoft Azure. This could be another avenue for further research.

References

- [1] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, Guillaume Lample: Unsupervised Translation of Programming Languages. In <https://arxiv.org/abs/2006.03511>
- [2] Lizi Ottens, Luis Perez, Sudharshan Viswanathan: Automatic Code Generation using Pre-Trained Language Models. In http://cs230.stanford.edu/projects_spring_2020/reports/38907662.pdf
- [3] Nazli Ugur Koyluoglu, Kaan Ertas, Austin Brotman: Pseudocode to Code Translation Using Transformers. In https://web.stanford.edu/class/cs224n/reports/final_reports/report100.pdf
- [4] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan: IntelliCode Compose: Code Generation using Transformer. In Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3368089.3417058>
- [5] Dan Hendrycks, Steven Basart et al: Measuring Coding Challenge Competence With APPS. In <https://arxiv.org/abs/2105.09938>
- [6] Mark Chen, Jerry Tworek, et al: Evaluating Large Language Models Trained on Code. In <https://arxiv.org/abs/2107.03374>
- [7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin: Attention is all you need. In <https://arxiv.org/abs/1706.03762>
- [8] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever: Language models are unsupervised multitask learners. In <http://www.persagen.com/files/misc/radford2019language.pdf>

A Appendix (optional)

A.1 Sample IaC: Virtual Machine in Azure defined via an ARM Template

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2021-11-01",
  "name": "[parameters('vmName')]",
  "location": "[parameters('location')]",
  "properties": {
    "hardwareProfile": {
```

```

    "vmSize": "[parameters('vmSize')]"
  },
  "storageProfile": {
    "osDisk": {
      "createOption": "FromImage",
      "managedDisk": {
        "storageAccountType": "[variables('osDiskType')]"
      }
    },
    "imageReference": {
      "publisher": "Canonical",
      "offer": "UbuntuServer",
      "sku": "[parameters('ubuntuOSVersion')]",
      "version": "latest"
    }
  },
  "networkProfile": {
    "networkInterfaces": [
      {
        "id": "[resourceId('Microsoft.Network/networkInterfaces', variables('networkInterfaceName'))]"
      }
    ]
  },
  "osProfile": {
    "computerName": "[parameters('vmName')]",
    "adminUsername": "[parameters('adminUsername')]",
    "adminPassword": "[parameters('adminPasswordOrKey')]",
    "linuxConfiguration": "[if(equals(parameters('authenticationType'), 'password'), null(), [parameters('linuxConfiguration')])]"
  },
  "dependsOn": [
    "[resourceId('Microsoft.Network/networkInterfaces', variables('networkInterfaceName'))]"
  ]
}

```