Fine-Tuning Large Language Model with Custom Dataset for Ansible Code Generation

Furkan Gürbüz

Technical University of Munich

TUM School of Computation, Information and Technology

Chair of Information Systems and Business Process Management (i17)

Prof. Dr. Stefanie Rinderle-Ma

Thomas Teubner, Noah Kim, Dr. Holger Wittges

München, 13. Mai 2025

Motivation

- Manual Ansible playbooks are complex and time-consuming
- Research Chair lacks of code generation LLM
- Sensitive information in prompts

Ansible Playbooks are like recipe books for servers

Key concepts of Ansible

- Written in YAML code
- Defines tasks
- playbook contains one or more sets of tasks

Simple Code snippet

```
- name: Say something
  hosts: localhost
  tasks:
    - name: Print a message
      debug:
        msg: "May the Force be with you”
```

RQ1: What methodology can be used to gather and prepare a custom dataset for fine-tuning large language models?

Methodology

- Used specialized tools to gather Ansible code samples
- Cleaned and preprocessed data to remove noise

Expected Result

- A clean, structured dataset ready for fine-tuning a large language model

RedHat‘s Ansible-Content-Parser

- Extracted code data from existing GitHub repositories
- Split the dataset into a ratio of 70/15/15 (train/val/test)


```
{"data_source_description": "",
"input": "---\n# Ansible Playbook for SAP NetWeaver (JAVA) with IBM Db2 Sandbox
installation\n\n# Use include_role / include_tasks inside Ansible Task block, instead of using
roles declaration or Task block with import_roles.\n# This ensures Ansible Roles, and the tasks
within, will be parsed in sequence instead of parsing at Playbook initialisation.\n\n\n####
Begin Infrastructure-as-Code provisioning ####\n\n- name: Ansible Play to gather input for
gathering vars and VM provisioning\n  hosts: localhost\n  gather_facts: false\n\n  # pre_tasks
used only for Interactive Prompts only and can be removed without impact\n  pre_tasks:\n\n    -
name: Playbook Interactive - Check if standard execution with an Ansible Extravars file is
requested by end user",
"license": "",
"module": "ansible.builtin.set_fact",
"output": "      ansible.builtin.set_fact:\n        playbook_enable_interactive_prompts: \"{{
true if (sap_vm_provision_iac_type is undefined and sap_vm_provision_iac_platform is undefined)
else false }}\"\n",
"path": "deploy_scenarios/sap_nwas_java_ibmdb2_sandbox/ansible_playbook.yml",
"repo_name": "ansible",
"repo_url": "https://github.com/sap-linuxlab/ansible.playbooks_for_sap"}
```

RQ2: What steps are involved in implementing the fine-tuning process for the large language model?

Methodology

- Selected a pre-trained LLM (Phi-4)
- Integrated LoRA for efficient fine-tuning
- Executed training with the custom Ansible dataset

Expected Result

- A fine-tuned LLM optimized for Ansible code generation in SAP environments

Microsoft‘s Phi-4 LLM can outperform larger LLM‘s

- 14 billion parameter **transformer-based** LLM
- architecture follows a **decoder-only** transformer

→ Reads prompt as part of the sequence, then **predicts** one token at a time

Benchmarks	Models						
	Phi-4 14B	**Phi-3** 14B	**Qwen 2.5** 14B instruct	**GPT** 4o-mini	**LLaMA-3.3** 70B instruct	**Qwen 2.5** 72B instruct	**GPT** 4o
MMLU	84.8	77.9	79.9	81.8	86.3	85.3	**88.1**
GPQA	**56.1**	31.2	42.9	40.9	49.0	50.6	50.6
MATH	**80.4**	44.6	75.6	73.0	66.3	74.6	74.6
HumanEval	82.6	67.8	72.1	86.2	78.9	87.1	**90.6**
MGSM	80.6	63.9	77.9	86.5	89.1	82.8	**90.4**
SimpleQA	3.0	7.6	7.6	39.4	9.3	8.6	9.3
DROP	75.5	58.3	59.7	79.9	82.4	80.9	**85.6**
MMLUPro	70.4	51.3	63.2	63.4	69.6	69.6	**73.0**
HumanEval+	82.8	69.2	79.1	82.4	77.8	84.0	**88.0**
ArenaHard	75.4	67.0	68.3	73.1	76.4	79.2	**85.6**
LiveBench	47.6	28.1	49.8	58.7	57.1	64.6	**72.4**
IFEval	63.0	57.9	78.7	78.7	**89.3**	85.6	84.8
PhiBench (internal)	56.2	43.9	49.8	58.7	57.1	64.6	**72.4**

RQ3: To what extend does the fine-tuned LLM meet the requirements?

Methodology

- Evaluated with ROUGE, METEOR, CHRF, and Ansible-lint
- Measured accuracy, syntax correctness, and code quality
- Iteration cycles from development to evaluation as defined by Hevner et al.

Expected Result

- Clear insights on how well the model performs in Ansible code generation tasks

Second Iteration metrics showcase significant increase in model performance

First Iteration Evaluation Scores	Second Iteration Evaluation Scores
'rouge1': np.float64(0.9030472630117133), 'rouge2': np.float64(0.8750916994262085), 'rougeL': np.float64(0.8937407874924173), 'rougeLsum': np.float64(0.8968060509310674)	'rouge1': np.float64(0.9089512020276276), 'rouge2': np.float64(0.8815313156096819), 'rougeL': np.float64(0.8997117333753821), 'rougeLsum': np.float64(0.9046659338280736)
'meteor': np.float64(0.8737622872894732)	'meteor': np.float64(0.8850545846138909)
'score': 97.2683986738109, 'char_order': 6, 'word_order': 0, 'beta’: 2	'score': 97.84404601871951, 'char_order': 6, 'word_order': 0, 'beta': 2
Overall Ansible Lint Score: 0.56	Overall Ansible Lint Score: 0.76

Limitations

- Dataset inherits **code quality & security issues** from public GitHub sources
- No **semantic and syntactical evaluation** by professional IaC developers

Future Outlook

- Embed the model in a **Fiori-based web application** on **SAP BTP**
- Deploy via **SAP AI Core** (side-by-side extensibility) or deploy via **FastAPI** or **Flask**, containerized with **Docker**
- Call **REST API service** from the frontend using **POST request**

Thank you for your attention!

Questions?