

QLoRA Fine-Tuning for Reliable Structured Tool Calls with GPT-2

Stanford CS224N Default Project

Jay Khemchandani

Department of Computer Science
Stanford University
jaykhem@stanford.edu

Abstract

My project is an extension of the CS224N default final project, in which I detail how I first implemented the core GPT-2 components and then the requisite downstream task pipelines. In this extension, I studied a new way to create structured data through tool calling. Out of the default project tasks, my final system scored 0.883 paraphrase accuracy on the Quora dev set and 0.883 paraphrase accuracy on the Quora test set. It also achieved a SacreBLEU CHRF score of 38.433 on the dev set and 40.600 on the test set. When it came to sentiment classification, my best saved dev results were 0.456 on SST using last-layer tuning and 0.861 on CFIMDB with full-model tuning. From that base level, I wanted to see whether fine-tuning can teach GPT-2 to generate schema-conforming tool calls that are valid given instructions in natural language. To do this, I constructed a synthetic dataset of 300 examples across 10 tools and formatted each example as a tool specification paired with a user request. I fine-tuned GPT-2 using QLoRA on the resulting prompt-completion pairs. I evaluated this using three strict metrics: JSON validity, adherence to schema, and exact match. When I prompted the base GPT-2 alone, I achieved 0 percent on all three metrics. In contrast, the QLoRA model trained on 255 examples reached 75.6 percent JSON validity, 57.8 percent schema adherence, and 8.9 percent exact match on a 45-example dev set. This project shows that even a 124M-parameter model can learn the structural form of tool use given some level of supervision, though it also highlights that extracting exact arguments is significantly harder than learning the format of the outputs alone.

1 Key Information

- Mentor: Shicheng Liu
- External collaborators: None
- External mentor: None
- Sharing project: No
- Late days: 1 late day remaining

2 Introduction

The default project for this class was to implement GPT-2 and adapt it to a few downstream tasks. In the first part of the project, I completed the GPT-2 components by adding masked self-attention transformer layers and the Adam optimizer, and then also implemented sentiment classification and downstream pipelines.

More concretely, the required implementation work from the default final project involved four main components. First, in `modules/attention.py`, I implemented causal multi-head self-attention

by projecting hidden states into queries, keys, and values, scaling the dot products, applying both the causal mask and the padding mask, normalizing with softmax, and combining the attended values across heads. Second, in `modules/gpt2_layer.py`, I implemented the GPT-2 block itself using residual connections, dropout, pre-layer normalization, and a GELU MLP. Third, in `models/gpt2.py`, I implemented the token-plus-position embedding stage and used that inside the full stacked transformer encoder, after which the model returns both the contextualized sequence output and the final non-padding token representation. Finally, in `optimizer.py`, I implemented AdamW with bias correction and decoupled weight decay, which was then used in the sentiment classifier training pipeline.

For the extension part, I wanted to study something very useful to me as a developer: creating structured tool calls. This is something I have noticed LLMs struggle with, even when using a multi-billion-parameter model from a frontier lab.

I was curious whether QLoRA could be a viable solution to this problem and create outputs in a machine-readable interface. I wanted to show that, if QLoRA, which is considerably computationally cheaper than retraining an entire model for a specific use case, could achieve exemplary results and perform significantly better than the baseline on a much smaller model, then that approach could be applied to bigger models and therefore lead to less developer friction and better outcomes for valuable real-life use cases.

Specifically, I wanted to implement structured tool calling where, given a natural language input and a tool specification, the model would return a JSON object with the correct tool name and arguments in the correct format. My core question was: can a parameter-efficient fine-tuning setup teach GPT-2 to reliably produce schema-conforming tool calls that could be used for valid API requests?

To do this, I first checked whether the model would output valid JSON structure. Then, if it did produce valid JSON, I checked whether it matched the correct schema for the specific tool. Further, I evaluated whether it could generate the exact intended arguments given the natural language input.

3 Related Work

While it is rare to see modern research being done on QLoRA or LoRA for small models today, it is still a fairly well-studied space, and there are other papers on this. In specific, I was directly motivated most by GPT-style autoregressive language modeling, tool-use fine-tuning for large language models, and more work on parameter-efficient adaptation. GPT-2, at least my implementation here, demonstrated that through next-token prediction alone, general-purpose language representations can be achieved with fairly high accuracy and efficiency Radford et al. [2019], and can even be used to generate sonnets, which was very impressive.

However, it was much worse at creating structured output that needs to satisfy formal constraints, which actually exist in real implementations. Recent work such as Gorilla studies how large language models can be trained to invoke APIs more reliably, though that work focuses on much larger models and richer tool corpora Patil et al. [2024].

My project is situated at a much smaller scale. But that is also very useful for applications where compute is not as readily available.

To make it even more feasible on limited compute, I used QLoRA, which is a quantized variant of LoRA (or low-rank adaptation), which is meant to freeze the base model to learn low-rank updates Hu et al. [2021], Dettmers et al. [2023]. QLoRA adds low-precision loading to reduce memory use Dettmers et al. [2023]. In my project, I did not just use QLoRA out of convenience, but it is also a part of my research question, since I wanted to see if parameter-efficient fine-tuning is a natural way to verify whether a small model can gain new structured learned behavior without necessitating a retraining of all of the weights of the model.

4 Approach

4.1 Task Formulation

Given a user instruction x and tool specification s , the model must generate a target tool call y :

$$p_{\theta}(y \mid x, s).$$

The target output is always a single JSON object with exactly two top-level keys: `name` and `arguments`. For example, if the instruction is to book a flight, the correct output might be a JSON object whose `name` is `book_flight` and whose `arguments` contain fields such as `origin`, `destination`, and `date`.

I cast this as standard causal language modeling. Each training example is a concatenation of a prompt and its target JSON completion. The training objective is next-token prediction over the entire sequence:

$$\mathcal{L}(\theta) = - \sum_{t=1}^T \log p_{\theta}(w_t \mid w_{<t}).$$

This is a simple formulation, but it is sufficient to test whether the model can internalize the structure of tool calling.

4.2 Base GPT-2 Implementation

Before working on the extension, I completed the core implementation steps required by the default project handout. The custom GPT-2 model uses learned word embeddings and learned positional embeddings, which are summed and passed through dropout before entering a stack of decoder-only transformer layers. Each transformer layer applies masked self-attention so that tokens cannot attend to future positions, followed by a feed-forward network, with residual connections and layer normalization around both sublayers. After the stack of layers, the model applies a final layer norm and extracts the last non-padding token representation for downstream tasks such as sentiment classification.

My paraphrase model in particular achieved 0.884 dev accuracy and 0.883 test accuracy, which was decent in absolute terms but only around median on the leaderboard. I think this reflects a limitation of the modeling choice more than a failure of the base GPT-2 implementation. In particular, I used a cloze-style prompt but trained GPT-2 with a binary classification head on the final token representation rather than truly generating the “yes” or “no” continuation. That made the system simple and efficient, but it likely underused the autoregressive structure of GPT-2 and limited how much sentence-pair information the model could exploit. I also kept the setup relatively lightweight, without stronger extensions such as larger models, ensembling, data augmentation, or more extensive hyperparameter tuning, so the result was a functioning but not especially optimized paraphrase system.

To match the implementation of a real GPT-2 model, in my code, pretrained Hugging Face weights are loaded into the custom architecture by remapping the original GPT-2 attention and MLP parameters into the corresponding linear layers in my implementation. This let me use the model both as a correctness-checked GPT-2 implementation for the base project and as the representation backbone for downstream tasks like sentiment classification, paraphrase detection, and sonnet generation.

4.3 Prompt Format

The prompt explicitly provides both the tool schema and the instruction. Concretely, each example is formatted as:

```
You are given a tool specification and a user instruction. Return a single JSON object with
exactly these keys: "name", "arguments". Do not output any other text.
TOOL_SPEC: {...}
INSTRUCTION: ...
```

This format is intentionally strict. It reduces ambiguity, makes the expected output contract explicit, and allows me to separate failures of structured generation from failures caused purely by underspecified prompts.

4.4 Model and Fine-Tuning Method

Although I made a custom GPT-2 implementation for the default project, the extension experiments use Hugging Face `AutoModelForCausalLM` with the 124M-parameter `gpt2` checkpoint for faster experimentation, and as I was having some issues in the beginning with creating a workable GPT-2 implementation that provided a limitation for how long I would be able to train my QLoRA model for the extension portion, I decided it would be better to use an implementation which was already effectively perfect. I fine-tuned this model with QLoRA. In the final run, the base model is loaded in 4-bit precision when the relevant libraries are available, and LoRA adapters are attached to the GPT-2 attention projection module `c_attn`. The final adapter settings are rank $r = 16$, scaling factor $\alpha = 32$, and dropout 0.05.

This design hopefully reflects the practical goal of the project. I am not trying to maximize raw model size; I am testing whether a small, efficient adaptation can inject structured tool-calling behavior into a pretrained model. Because the model is still trained with ordinary causal language modeling, any improvements can be attributed primarily to the fine-tuning data and the adapter-based update mechanism rather than to a highly specialized decoding algorithm.

5 Experiments

5.1 Default Project Results

On paraphrase detection, my evaluation run for `10-1e-05-paraphrase.pt` achieved 0.884 accuracy on the Quora dev set and generated the final dev and test prediction files. The eventual leaderboard results were 0.884 on dev and 0.883 on test. On sonnet generation, the final autograded scores were 38.433 CHRF on dev and 40.600 CHRF on test.

For sentiment classification, with `-fine-tune-mode last-linear-layer`, GPT-2 achieved 0.456 SST dev accuracy and 0.857 CFIMDB dev accuracy. With `-fine-tune-mode full-model`, it achieved 0.400 SST dev accuracy and 0.861 CFIMDB dev accuracy. This suggests that full-model tuning slightly helped on CFIMDB but overfit on SST, where training accuracy reportedly reached 0.981 while dev accuracy remained low. This overfitting pattern makes sense, as SST is smaller and more fine-grained than CFIMDB.

5.2 Data

I generated a synthetic tool-calling dataset using ChatGPT which covers 10 tools: weather lookup, flight booking, email sending, reminders, restaurant search, translation, alarms, stock lookup, currency conversion, and tip calculation. Each tool comes with a JSON-style schema, multiple natural-language templates, and randomized slot values. I created 300 examples total, approximately balanced across tools, while filtering out trivial invalid cases such as booking a flight from and to the same location.

After generation, I normalized the data, removed ambiguous or malformed examples, created more in their place, and split the resulting set into 255 training examples and 45 development examples using seed 11711. The tools vary meaningfully in difficulty. Some require only one argument, while others require three. Some arguments are strings, while others are numeric. Some tools also include optional arguments, which served to make exact recovery harder even when the model learns the right overall format.

5.3 Evaluation Method

I evaluated the model with three metrics of increasing strictness.

JSON validity measures whether the decoded output can be parsed as JSON. This is the minimal structural requirement.

Schema adherence measures whether the parsed JSON has the expected top-level structure, uses the correct tool name, includes required arguments, avoids unknown arguments, and assigns arguments with the correct coarse JSON types.

Exact match requires the tool name and all arguments to exactly match the gold tool call after canonicalizing argument order. This is the strictest metric because the output must be both well formed and semantically correct.

This metric suite is important for interpreting results. A model may learn how a tool call should *look* before it learns how to recover the exact values inside that call.

5.4 Experimental Details

My primary baseline is prompt-only GPT-2, which uses the same prompt format but performs no weight updates. I also ran a small-data QLoRA experiment using only 14 training examples to probe whether attaching adapters alone is enough to induce structure. The final model uses the full 255-example training split, learning rate 3×10^{-4} , batch size 4, maximum sequence length 384, and 5 epochs of training. This corresponds to 320 optimizer steps in the saved run. All experiments use seed 11711.

Training is intentionally simple. The implementation tokenizes the concatenated prompt and gold JSON, sets labels equal to the input IDs, and fine-tunes with Hugging Face Trainer. I do not apply completion-only loss masking, curriculum learning, constrained decoding, or explicit validation-time model selection. At inference time, I use greedy decoding with a fixed maximum generation length. This makes the final results a clean test of what basic QLoRA fine-tuning can achieve on its own.

5.5 Results

Table 1 shows the main development-set results.

Method	JSON Validity	Schema Adherence	Exact Match
Prompt-only GPT-2	0.0%	0.0%	0.0%
QLoRA (14 train)	0.0%	0.0%	0.0%
QLoRA (255 train)	75.6%	57.8%	8.9%

Table 1: Development set results. The final QLoRA model corresponds to 34/45 valid JSON outputs, 26/45 schema-adherent outputs, and 4/45 exact matches.

The most important result here is that QLoRA dramatically improved the structural correctness in terms of the JSON output, where prompt-only GPT-2 failed completely. It often copied the prompt, produced explanatory text, or emitted truncated schema fragments instead of a JSON object. The small-data QLoRA run also remains at 0%, suggesting that simply adding adapters is not enough; the model needs a nontrivial amount of structured supervision. Once trained on 255 examples, however, the model produces valid JSON on most examples and schema-adherent outputs on over half of the dev set.

At the same time, exact match remains low. This means the model has learned a great deal about *format* and partial argument structure, but not yet the full semantics needed for precise slot extraction. In other words, structured fine-tuning is clearly helping, but the hardest part of the task has shifted from syntax to semantic correctness.

6 Analysis

The overall pattern in the results that I got shows a separation between semantic reasoning and structural learning. The final model was able to output something that looked like a correct tool call, but there were often a few fields that were incorrect, which of course would result in actual API failures in production. It is still encouraging since it shows that my model did learn the high-level protocol to an exponentially greater degree than the prompt-only GPT-2 baseline, which I found was really, really interesting.

However, there is still a lot to go in terms of this project, as there were a few precise failure modalities. For example:

1. Some were still invalid JSON outputs due to missing commas or braces.

2. The model often emitted the correct field names and used string values where numbers were expected, which is fairly understandable.

Sometimes the right tool was used, but not necessarily the right semantics. It could have swapped currencies in currency conversion or misrepresented what a user actually wanted.

In general, this shows that the model is getting the shape of the desired output correctly, but it is not perfect yet, which reminds me a bit of a diffusion model, which does make a lot of sense.

Going from 14 training examples to 255 examples changes the model from complete failure to mostly well-formed generation. That suggests the extension is not merely exploiting memorization of a handful of examples; rather, the model is learning a general structural prior over the task. Even so, 255 examples is still small for a problem that requires both structural consistency and accurate information extraction across 10 different tools.

7 Conclusion

Overall, with this project, I wanted to see whether QLoRA fine-tuning could actually teach GPT-2 to produce reliable, structured tool calls. I would say that this project was a resounding success, but it is not done yet.

Clearly, we saw a lot of improvement from the baseline. However, the model still had significant issues.

My next steps would be:

1. Create a completion-only loss masker.
2. Create a dataset that is larger and more diverse.
3. Train the model on that and run comparisons against standard LoRA or full fine-tuning.

Of course, switching to a larger parameter model would also surely help. I wonder if doing this with a model that is perhaps three or four times larger would produce significantly better results, though I would not necessarily want to make this experiment on a model which is too large and well-developed, as that ignores the benefit of this being lightweight and somewhat usable on consumer hardware.

Team Contributions

This was a single-author project. I completed the GPT-2 implementation work for the default project, designed the tool-calling extension, generated the synthetic dataset, implemented the QLoRA training and evaluation pipeline, ran the experiments, and wrote the report.

References

- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023. URL <https://arxiv.org/abs/2305.14314>.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. URL <https://arxiv.org/abs/2106.09685>.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

A Appendix

A.1 Prompt Template

You are given a tool specification and a user instruction.
Return a single JSON object with exactly these keys: "name", "arguments".
Do not output any other text.

TOOL_SPEC:
{tool_schema}

INSTRUCTION:
{instruction}

A.2 Representative Error Modes

Three representative failure modes from the saved dev predictions are:

- **Invalid JSON:** missing commas or braces, often after partially correct argument extraction.
- **Wrong argument types or missing arguments:** especially on numeric tools such as tip calculation.
- **Semantically incorrect but well-formed outputs:** for example, the model selects the correct tool and fields but swaps source/target roles or copies extra words from the instruction into an argument.