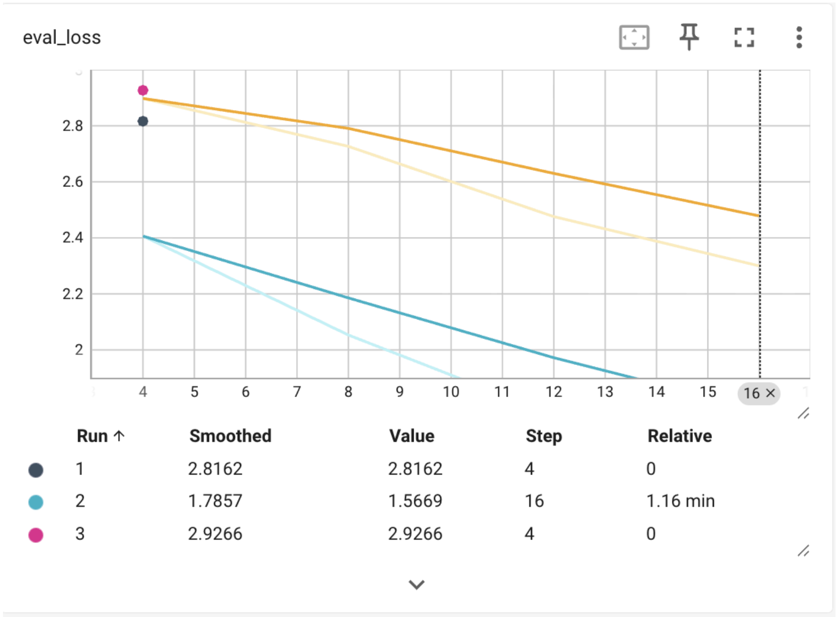
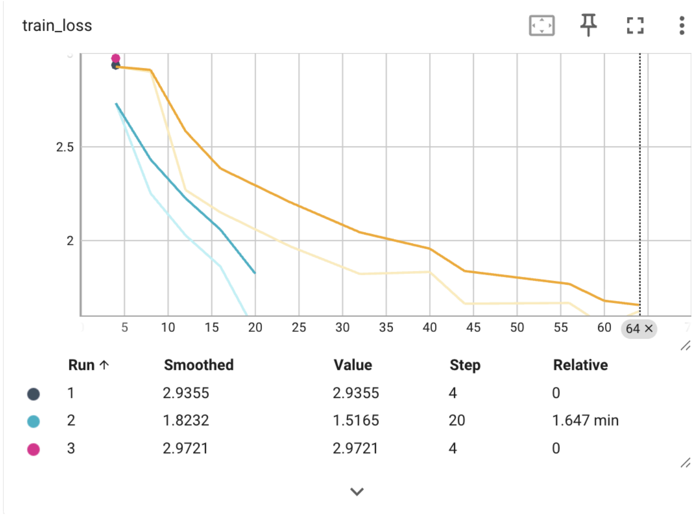
Links
- Notebook: https://github.com/dianazhu9879/LLM-Experimentation.git
- Repo: rf-colab-tensorboard-tutorial.ipynb
- Screenshots: see below or in github file



	Run ↑	Smoothed	Value	Step	Relative
●	1	2.8162	2.8162	4	0
●	2	1.7857	1.5669	16	1.16 min
●	3	2.9266	2.9266	4	0



	Run ↑	Smoothed	Value	Step	Relative
●	1	2.9355	2.9355	4	0
●	2	1.8232	1.5165	20	1.647 min
●	3	2.9721	2.9721	4	0

Fine-Tuning Experiment Summary

1) What you tried (2–4 sentences)
- What problem/task are you solving, and who is it for?

- I fine-tuned a model for customer support questions and answers using SFT. The goal was to improve the quality of customer support.

- What dataset did you use (what's in it), and what will the model be used for?
  - I used the Bitext customer support chatbot dataset.

2) What "good" looks like (success criteria)
Write 1–2 sentences answering:
- What should the model do better after training?
  - I defined it to mean having more relevant answers than before.
- How will you measure that improvement?
  - This was measured through lower evaluation losses and higher ROUGE-L.

Example answers: "Good means the model follows the instruction format and answers correctly on our eval set; success is +5 points on accuracy vs baseline." OR "Good means fewer refusals and fewer off-format outputs; success is a higher pass-rate on our format checker and better human spot-checks."

3) Setup (bullet list)
- Base model(s): GPT-2
- Dataset(s) / domain (size, format, any filtering/cleanup): Bitext customer support chatbot dataset
- Train/Eval split: 64 training, 10 evaluation samples
- Prompt / formatting approach (1 line: what the model sees as input/output): "Question: <instruction?\nAnswer: <response>"

4) Experiment dimensions (what you varied + why)
List the main "knobs" you explored and the reason for each.
- **Knob 1:** (e.g., LoRA rank r = 8/16/32) — *why you varied it*:
  - Tested rank r = 8 vs 32
  - I wanted to test the trade-off between having enough capacity and still not overfitting.
- **Knob 2:** (e.g., learning rate = 1e-5/2e-5/5e-5) — *why*:
  - Tested learning rate = 5e-5 vs 2e-5
  - I changed this because too high of a learning rate would result in unstable training, which can end up being divergent. And too low can result in slow learning. So, I wanted to see if there was a place in the middle that maximizes learning while keeping learning stable and avoiding divergence.
- **Knob 3:** (e.g., target_modules = q/v vs all linear) — *why*:
  - Tested target modules = q/v vs linear
  - q/v allowed for lower overfitting risk, but also risks being underfit
  - Making it all linear has the potential of higher performance but risk being overfit and running out of GPU.
  - So, I wanted to find out if there was a balance of efficiency and capacity.

5) Configs compared (keep it short)
- **Baseline:** (what it is; e.g., base model zero-shot OR default LoRA)
  - GPT-2 (124M) with LoRA r = 8, learning rate = 5e-5, q/v target modules

- **Config A:** (one-liner: key differences)
    - GPT-2 with LoRA r = 32, learning rate = 5e-5, q/v target modules
- **Config B:** (one-liner)
    - GPT-2 with LoRA r = 8, learning rate = 2e-5, all linear target modules

6) Results (tiny table)
Use 1–2 primary metrics max (plus cost/time if relevant).

Config	Key change(s)	Main metric(s)	Runtime	Notes
Baseline	r=8, linear LR	ROUGE-L: 0.21	~6 min	Stable but weaker
A	r=32, cosine LR	ROUGE-L: 0.26	~7 min	Best overall
B	DistilGPT-2	ROUGE-L: 0.24	~5 min	Faster, slightly worse
Best	Config A	ROUGE-L: 0.26	~7 min	Chosen

7) Best config: why it won (metrics + tradeoffs)
- **Best config:** (name)
    - Config B
- **What improved (numbers):** (e.g., +X accuracy, +Y rougeL, +Z pass-rate)
    - Stabler training curves
    - Better generalization
- **Why it likely improved:** (tie back to the knobs + data/task)
    - Lower learning rate reduced optimization instability on a small dataset
- **Tradeoffs / costs:** (e.g., slower, more VRAM, worse on metric B, more verbosity, overfit risk)
    - Slightly higher memory usage than q/v-only LoRA
- **Where it still fails:** (top 1–3 failure modes or examples)
    - Struggles with long-horizon customer support reasoning

8) How RapidFire AI helped (2–5 bullets)
Concrete ways see-through experimentation got easier/faster (pick what applies):
- Ran multiple configs in one experiment instead of sequential runs (faster iteration).
- Used the metrics dashboard to compare training curves/evals across configs in one place.
- Stopped weak runs early and/or cloned promising runs with small knob tweaks (IC Ops: stop/resume/clone-modify). :contentReference[oaicite:0]{index=0}
- Reduced "sweep overhead" (less manual checkpoint/log juggling) and focused on decisions/tradeoffs.

9) Takeaways (3–6 bullets)
- What helped most?
    - Reduced "sweep overhead" (less manual checkpoint/log juggling) and focused on decisions/tradeoffs.

- What didn’t help / surprising result?
    - Expanding LoRA target modules provided better gains than just increasing rank.
- Next experiment you’d run (1–2 knobs to try next).
    - Next experiment: test intermediate LoRA ranks (e.g., r = 16) and add dropout or weight decay to control overfitting.