🎓 RLHF Poem Sentiment Demo v2

Reinforcement Learning from Human Feedback on Classic Poetry Continuation

- **Base Model**: `meta-llama/Llama-3.2-1B-Instruct` (Parameter-Efficient with LoRA)
- **Dataset**: `PoetryDB` (Classic English Poetry: Shakespeare, Byron, Dickinson, Keats, Shelley)
- **Reward Signal**: Frozen CardiffNLP RoBERTa sentiment classifier
- **Optimization Algorithm**: Group Relative Policy Optimization (GRPO) with a KL drift penalty

---

💡 Core Question for Today's Class

In Supervised Fine-Tuning (SFT), a model learns to imitate a data distribution. But in real-world alignment (e.g., helpfulness, safety, tone steering), we want to **guide** behavior using a scalar reward signal.

- **Task**: Given the first four lines of a classic poem, write four additional lines.
- **Reward**: The frozen classifier scores the complete reply using $P(\text{positive}) - P(\text{negative})$.
- **Goal**: Watch the model learn in real time to steer its output toward joy, warmth, and uplifting themes while preserving poetic meter and rhyming rhythm.

0. Environment & Hardware Check

Verify GPU availability and package versions (`torch` , `transformers` , `peft` , `trl`).

In [1]:
```
import torch
import transformers
import peft
import trl

print(f"PyTorch Version:      {torch.__version__}")
print(f"Transformers Version: {transformers.__version__}")
print(f"PEFT Version:         {peft.__version__}")
print(f"TRL Version:          {trl.__version__}")
```



```
print(f"CUDA Available:       {torch.cuda.is_available()}")
print(f"GPU Count:            {torch.cuda.device_count()}")
for i in range(torch.cuda.device_count()):
    props = torch.cuda.get_device_properties(i)
    print(f"  GPU {i}: {props.name} ({props.total_memory / 1e9:.1f} GB VRAM)")
```

```
PyTorch Version:      2.8.0+cu128
Transformers Version: 5.20.1
PEFT Version:         0.22.0
TRL Version:          1.18.1
CUDA Available:       True
GPU Count:            2
  GPU 0: NVIDIA RTX A6000 (50.9 GB VRAM)
  GPU 1: NVIDIA RTX A6000 (50.9 GB VRAM)
```

1. Exploring the PoetryDB Dataset

The dataset contains classic poems with at least 16 lines:

- **Prompt**: The first four lines.
- **Reference Continuation**: The next 4 lines written by the original poet (the ground truth).

In [2]:
```
from data_utils import load_poems, slice_poem
from demo_config import POEM_PROMPT_LINES
import random

# Load curated poems
poems = load_poems()
print(f"Loaded {len(poems)} curated poems from PoetryDB.\n")

# Randomly sample and inspect a sliced poem
sample_poem = random.choice(poems)
sliced = slice_poem(sample_poem, num_prompt_lines=POEM_PROMPT_LINES)

print(f"Title:  {sliced['title']}")
print(f"Author: {sliced['author']}")
print("~" * 55)
print(f"--- PROMPT (First {sliced['num_prompt_lines']} lines) ---")
print(sliced['prompt_text'])
print("~" * 55)
```



```
print("--- ORIGINAL CONTINUATION (Poet's next 4 lines) ---")
print(sliced['reference_text'])
print("~" * 55)
```

```
Loaded 553 curated poems from PoetryDB.

Title:  Elegiac Stanzas on the Death of Sir Peter Parker, Bart
Author: George Gordon, Lord Byron
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
--- PROMPT (First 4 lines) ---
There is a tear for all that die,
A mourner o'er the humblest grave;
But nations swell the funeral cry,
And Triumph weeps above the brave.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
--- ORIGINAL CONTINUATION (Poet's next 4 lines) ---
For them is Sorrow's purest sigh
O'er Ocean's heaving bosom sent:
In vain their bones unburied lie,
All earth becomes their monument!
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

2. The reward model

`cardiffnlp/twitter-roberta-base-sentiment-latest` estimates positive, neutral, and negative sentiment. The reward is the positive probability minus the negative probability. Empty replies receive -1.

Read the continuations for coherence and relevance. The sentiment classifier can give an off-topic reply a high score.

In [3]:
```
from reward_utils import SentimentRewardModel

# Initialize Sentiment Reward Model (uses second GPU if available)
reward_model = SentimentRewardModel()

# Test the reward model on 3 contrasting poetic completions
test_verses = [
    # Verse 1: Joyful / Radiant
    "The golden sun smiles warmly from on high,\nAnd fills the joyful meadow with delight;\nSweet songs of

    # Verse 2: Melancholic / Somber
```



```
    "A cold dark grave where sorrow ever weeps,\nAnd dreary shadows cloak the haunted ground;\nIn bitter gl

    # Verse 3: Short Fragment
    "The trees were green."
]

scores = reward_model(completions=test_verses)

print("Reward Model Evaluation:")
print(f"  1. Joyful verse       {scores[0]:+.4f}")
print(f"  2. Melancholic verse  {scores[1]:+.4f}")
print(f"  3. Short fragment     {scores[2]:+.4f}")
```

```
Loading Sentiment Reward Model (cardiffnlp/twitter-roberta-base-sentiment-latest) on cuda:1...
Loading weights:   0%|          | 0/201 [00:00<?, ?it/s]
[transformers] RobertaForSequenceClassification LOAD REPORT from: cardiffnlp/twitter-roberta-base-sentimen
t-latest
Key                          | Status     |  |
-----------------------------+------------+--+-
roberta.pooler.dense.bias    | UNEXPECTED |  |
roberta.pooler.dense.weight | UNEXPECTED |  |

Notes:
- UNEXPECTED:   can be ignored when loading from different task/architecture; not ok if you expect identica
l arch.
Sentiment Reward Model loaded successfully.
Reward Model Evaluation:
  1. Joyful verse       +0.9592
  2. Melancholic verse  -0.7598
  3. Short fragment     +0.7582
```

3. Compare two KL weights

Each run trains a fresh LoRA adapter from Llama-3.2-1B-Instruct using the same four-line chat prompts, training split, seed, and reward function. The live run uses 30 updates per beta. The prepared experiment in `rlhf-happy` used 300 updates per condition.

GRPO adds β times the estimated per-token KL divergence from the frozen base policy to the loss. For the same KL divergence, $\beta = 0.15$ contributes three times the penalty of $\beta = 0.05$.

The dashboard shows the active run's reward, loss, and KL. The following plots overlay both runs so they can be compared on the same scales.

In [4]:
```
from demo_config import LIVE_ADAPTER_PATH
from live_monitor import LiveTrainingVisualizer
from train_rlhf import train

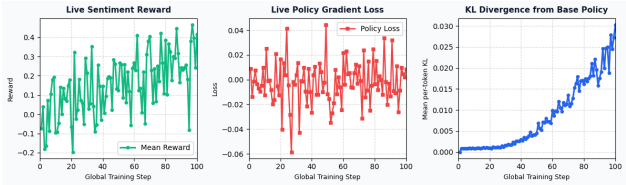
NUM_LIVE_STEPS = 100
BETAS = (0.05, 0.15)
beta_labels = {beta: f"β = {beta:.2f}" for beta in BETAS}
adapter_names = {beta: f"beta_{beta:.2f}".replace(".", "_") for beta in BETAS}
adapter_paths = {beta: f"{LIVE_ADAPTER_PATH}_beta_{beta:.2f}" for beta in BETAS}
training_runs = {}

for beta in BETAS:
    visualizer = LiveTrainingVisualizer(total_steps=NUM_LIVE_STEPS, update_interval=3, beta=beta)
    reward_model.on_sample_hook = visualizer.on_sample_hook
    train(
        max_steps=NUM_LIVE_STEPS,
        output_dir=adapter_paths[beta],
        beta=beta,
        reward_model=reward_model,
        callbacks=[visualizer],
        logging_steps=1,
        save_strategy="no",
    )
    training_runs[beta] = visualizer

reward_model.on_sample_hook = None
```

🚀 Live RLHF Policy Training Progress: Step 100 / 100 (100%)

GRPO with sentiment reward and KL regularization (β = 0.15)



📝 Latest Generated Rollout Sample **Sentiment Reward: 0.4559**

Weave close, just you and I
With ropes that tangle tight and fly
Late night sunbeams dry the rain
On windy days your wild heart's again

```
Training complete! Saving LoRA adapter to /home/canders1/rlhf-s1/v2/checkpoints/chat/live_demo_adapter_beta
_0.15...
Done!
```

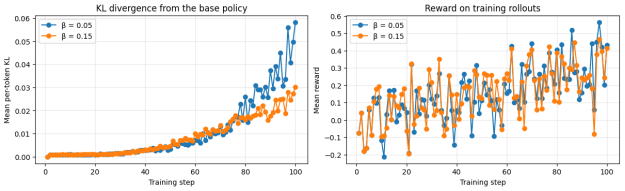
In [11]:
```
import matplotlib.pyplot as plt

beta_colors = {beta: plt.get_cmap("tab10")(i % 10) for i, beta in enumerate(BETAS)}
fig, (kl_ax, reward_ax) = plt.subplots(1, 2, figsize=(12, 3.6), layout="constrained")

for beta, run in training_runs.items():
    kl_ax.plot(run.steps, run.kl_values, marker="o", color=beta_colors[beta], label=beta_labels[beta])
    reward_ax.plot(run.steps, run.rewards, marker="o", color=beta_colors[beta], label=beta_labels[beta])

kl_ax.set(title="KL divergence from the base policy", ylabel="Mean per-token KL")
reward_ax.set(title="Reward on training rollouts", ylabel="Mean reward")
```

```
for ax in (kl_ax, reward_ax):
    ax.set_xlabel("Training step")
    ax.grid(alpha=0.3)
    ax.legend()
plt.show()
```



4. Compare continuations on one poem

Both adapters and the base model get the same held-out prompt and sampling seed. Evaluation uses the same chat template and decoding settings as training. The comparison shows each full reply and its sentiment score.

In [5]:
```
from data_utils import prepare_rlhf_datasets
from evaluate_completions import PoemEvaluator
from demo_ui import display_poem_comparison

_, _, _, held_out_poems = prepare_rlhf_datasets()
EVAL_SEED = 42

first_beta = BETAS[0]
evaluator = PoemEvaluator(
    adapter_path=adapter_paths[first_beta],
    adapter_name=adapter_names[first_beta],
)
for beta in BETAS[1:]:
```



```
    evaluator.peft_model.load_adapter(adapter_paths[beta], adapter_name=adapter_names[beta])
evaluation_adapters = {beta_labels[beta]: adapter_names[beta] for beta in BETAS}
```

```
Loading Base Model (meta-llama/Llama-3.2-1B-Instruct) on cuda:0...
[transformers] `torch_dtype` is deprecated! Use `dtype` instead!
Loading weights:   0%|          | 0/146 [00:00<?, ?it/s]
Loading LoRA RLHF Adapter from /home/canders1/rlhf-s1/v2/checkpoints/chat/live_demo_adapter_beta_0.05...
Loading Sentiment Reward Model (cardiffnlp/twitter-roberta-base-sentiment-latest) on cuda:1...
Loading weights:   0%|          | 0/201 [00:00<?, ?it/s]
[transformers] RobertaForSequenceClassification LOAD REPORT from: cardiffnlp/twitter-roberta-base-sentimen
t-latest
Key                           | Status     |  |
------------------------------+------------+--+-
roberta.pooler.dense.bias     | UNEXPECTED |  |
roberta.pooler.dense.weight   | UNEXPECTED |  |

Notes:
- UNEXPECTED:   can be ignored when loading from different task/architecture; not ok if you expect identica
l arch.
Sentiment Reward Model loaded successfully.
All models ready for evaluation.
```

In [9]:
```
# Rerun this cell to sample another held-out poem.
result = evaluator.compare_adapters(random.choice(held_out_poems), evaluation_adapters, seed=EVAL_SEED)
display_poem_comparison(result)
```


Bigotry's Victim

By Percy Bysshe Shelley

Prompt · First 4 lines

Dares the lama, most fleet of the sons of the wind,
The lion to rouse from his skull-covered lair?
When the tiger approaches can the fast-fleeting hind
Repose trust in his footsteps of air?

Original Poet Reward -0.808

No! Abandoned he sinks in a trance of despair,
The monster transfixes his prey,
On the sand flows his life-blood away;
Whilst India's rocks to his death-yells reply,

Base Model Reward -0.001

Its call echoes through the verdant, grassy plain,
And all in slumber tremble with mystic pain,
The eagle's soft whisper lies on distant shore,
As Solène's eyes drift with nocturnal lore,
And in Lha-Ong's realm, its mystic touch implores.

β = 0.05 Reward 0.521

Its call echoes through the verdant, grassy herd,
His heart afire with a nomadic proud creed,
The eagle soars, its wings outstretched and wide,
Unsetting strategy with an effortless deed.

β = 0.15 Reward 0.012

Its call echoes through the verdant, grassy hermitage.

Would the stallion, swift and proud, be afraid?
The softest breeze whispers secrets of the trade.
The eagle soars with only noble purpose.
And in its absence, doth the deer claim.

5. Compare rewards on held-out poems

Compare sentiment and four-line compliance on six held-out prompts. Four-line compliance counts nonempty lines in the full reply.

In [12]:
```
import pandas as pd

NUM_TEST_POEMS = 6
test_poems = random.Random(EVAL_SEED).sample(held_out_poems, NUM_TEST_POEMS)
reward_columns = ["Original Poet", "Base Model", *evaluation_adapters]
batch_results = []
four_line_counts = dict.fromkeys(reward_columns, 0)
for i, poem in enumerate(test_poems):
    res = evaluator.compare_adapters(poem, evaluation_adapters, seed=EVAL_SEED + i)
    for label, entry in res["continuations"].items():
        four_line_counts[label] += sum(bool(line.strip()) for line in entry["text"].splitlines()) == 4
    batch_results.append({
        "Poem Title": res["title"],
        "Author": res["author"],
```



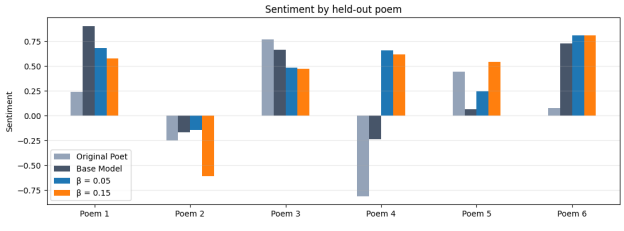
```
        **{label: entry["reward"] for label, entry in res["continuations"].items()},
    })

df = pd.DataFrame(batch_results)
display(df.round(3))
summary = df[reward_columns].mean().to_frame("Mean sentiment")
summary["Four-line replies (%)"] = 100 * pd.Series(four_line_counts) / NUM_TEST_POEMS
display(summary.round(3))

ax = df[reward_columns].plot.bar(
    figsize=(11, 4),
    color=["#94a3b8", "#475569", *(beta_colors[beta] for beta in BETAS)],
    rot=0,
)
ax.set_xticks(range(len(df)), [f"Poem {i+1}" for i in range(len(df))])
ax.set(title="Sentiment by held-out poem", ylabel="Sentiment")
ax.grid(axis="y", alpha=0.3)
ax.figure.tight_layout()
plt.show()
```

	Poem Title	Author	Original Poet	Base Model	β = 0.05	β = 0.15
0	I'll tell you how the sun rose,	Emily Dickinson	0.237	0.901	0.680	0.577
1	The wind begun to rock the grass	Emily Dickinson	-0.247	-0.168	-0.142	-0.610
2	Egotism. A Letter to J. T. Becher	George Gordon, Lord Byron	0.765	0.661	0.481	0.472
3	Bigotry's Victim	Percy Bysshe Shelley	-0.808	-0.239	0.655	0.619
4	Because I could not stop for Death	Emily Dickinson	0.445	0.066	0.243	0.544
5	One day is there of the series	Emily Dickinson	0.078	0.729	0.806	0.806

	Mean sentiment	Four-line replies (%)
Original Poet	0.078	100.000
Base Model	0.325	100.000
β = 0.05	0.454	100.000
β = 0.15	0.401	83.333



6. Classroom Discussion Questions 🧠

1. The Alignment vs. Authenticity Dilemma

- Classical poetry often dwells in tragedy, heartbreak, and death.
- When presented with a melancholy prompt, how do the two adapters and the base model handle the original tone?

2. Reward Hacking & Policy Drift

- How does the KL penalty change the incentive to output a string of positive words such as `"happy joy joyful bliss smile sunshine love gladness"`?
- Which run had lower KL on its training rollouts? How did held-out rewards and poem quality compare? Consider how the short run and small sample affect your conclusions.

3. Why GRPO over Traditional PPO?

- In classic PPO, we must initialize and train a separate **Critic (Value) Network** of the same scale as the language model to estimate state values $V(s)$.

- In DeepSeek's **GRPO**, the model generates G completions per prompt and calculates relative advantages:

$$A_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$$

- This eliminates the memory and instability of a separate value network, making online RLHF accessible and fast!