RLHF In-Class Demo

Sentiment-Steered Poem Completion with GRPO

Llama 3.2 1B · PoetryDB · DistilBERT SST-2 reward · Group Relative Policy Optimization

---

Section 0 — Setup

In [1]:
```
import json
import random
import subprocess
import sys
from collections import deque
from html import escape
from pathlib import Path

import ipywidgets as widgets
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import requests
from IPython.display import Image, clear_output, display

# Jupyter can start the kernel from notebooks/ or the project root.
working_dir = Path.cwd()
PROJECT_ROOT = next(
    (p for p in (working_dir, *working_dir.parents)
     if (p / 'training' / 'train_grpo.py').is_file()),
    None,
)
if PROJECT_ROOT is None:
    raise RuntimeError('Start the notebook from the project root or notebook
if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))

from data.poem_loader import POETRYDB_BASE, build_prompt

print('Libraries loaded.')
```

```
Libraries loaded.
```

In [2]:
```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

print(f'PyTorch: {torch.__version__}')
print(f'CUDA available: {torch.cuda.is_available()}')
for i in range(torch.cuda.device_count()):
    p = torch.cuda.get_device_properties(i)
```

```
    print(f'  GPU {i}: {p.name}  ({p.total_memory // 1024**3} GB)')
```

```
PyTorch: 2.8.0+cu128
CUDA available: True
  GPU 0: NVIDIA RTX A6000  (47 GB)
  GPU 1: NVIDIA RTX A6000  (47 GB)
```

In [3]:
```
BASE_MODEL_ID = 'meta-llama/Llama-3.2-1B-Instruct'
MODEL_DTYPE = (
    torch.bfloat16 if torch.cuda.is_available() and torch.cuda.is_bf16_suppo
    else torch.float32
)
CHECKPOINT_ROOT = PROJECT_ROOT / 'checkpoints'
CHECKPOINT_LOW = CHECKPOINT_ROOT / 'beta_0_04'
CHECKPOINT_HIGH = CHECKPOINT_ROOT / 'beta_0_4'
TRAIN_MAX_STEPS = 100  # Per beta run.

print('Config set.')
```

```
Config set.
```

In [4]:
```
# ── Load base model (used for reference inference & perplexity) ──────────
print('Loading tokenizer...')
tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_ID)
tokenizer.pad_token = tokenizer.eos_token

print('Loading base model...')
base_model = AutoModelForCausalLM.from_pretrained(
    BASE_MODEL_ID,
    dtype=MODEL_DTYPE,
    device_map='auto',
)
base_model.eval()
print('Base model ready.')
```

```
Loading tokenizer...
Loading base model...
Loading weights:   0%|          | 0/146 [00:00<?, ?it/s]
Base model ready.
```

In [5]:
```
from training.reward import score_text

print('Sentiment model ready.')
```

```
Loading weights:   0%|          | 0/104 [00:00<?, ?it/s]
[reward.py] Sentiment model loaded on device=0
Sentiment model ready.
```

Section 1 — RLHF & GRPO Concepts

What is RLHF?

Reinforcement Learning from Human Feedback aligns a language model's outputs toward a desired behavior by training it with a reward signal instead of supervised labels.

Standard RLHF pipeline (PPO)

```
[prompts] → policy → completions → reward model → scalar
reward → PPO update
                ↑_____________________KL penalty (vs reference
policy)___________|
```

Problems: requires a trained reward model, a value network, and a reference model — 4 models in memory.

GRPO (Group Relative Policy Optimization)

```
[prompt] → generate G completions → reward_fn(each) →
normalize within group → policy gradient
```

Key insight: Instead of a learned value baseline, GRPO uses the **mean reward of the group** as the baseline:

$$A_i = \frac{r_i - \mu_G}{\sigma_G} \quad \text{where } G = \{r_1, \ldots, r_8\} \text{ for the same prompt}$$

No value network. No reward model training. The reward function can be **any callable** — including a pretrained sentiment classifier.

Our reward function

$$R = \underbrace{R_{\text{sentiment}}}_{\text{SST-2 confidence} \in [-1,1]} + 0.2 \times \underbrace{R_{\text{structure}}}_{\mathbb{1}[\text{3–5 newlines}]}$$

The KL penalty β

$$\mathcal{L} = -\mathbb{E}[A_i \log \pi_\theta(y|x)] + \beta \cdot D_{KL}(\pi_\theta || \pi_{\text{ref}})$$

β	Effect
0.04 (low)	Model drifts far from base; strong sentiment shift, may lose poetic coherence
0.4 (high)	Stays close to base distribution; modest sentiment shift, more coherent poetry

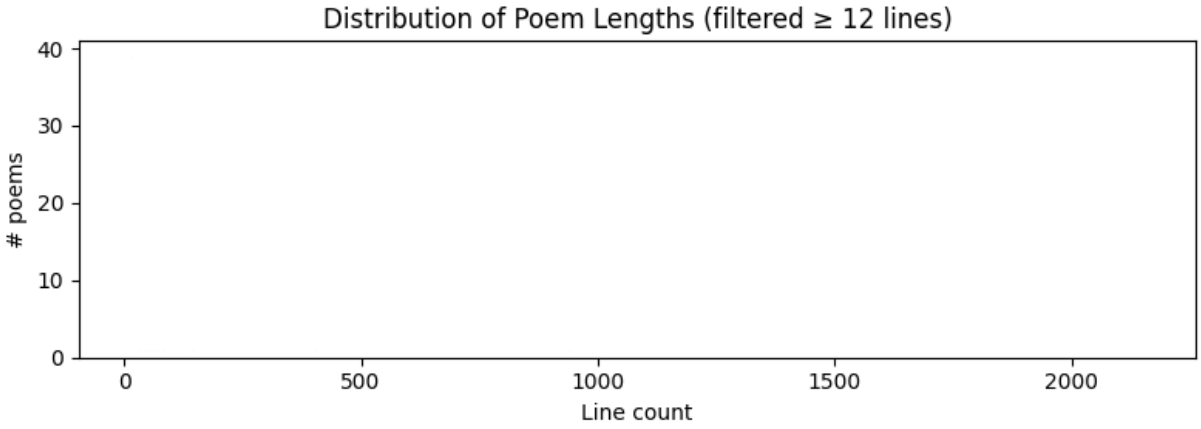
> **Demo goal:** Show that the reward signal — not the prompt — drives the sentiment shift. The system prompt is completely neutral.

Section 2 — PoetryDB Dataset Explorer

In [6]:
```
# ── Fetch poems from PoetryDB ──────────────────────────────────────
print('Fetching poems from poetrydb.org...')
resp = requests.get(f'{POETRYDB_BASE}/random/200', timeout=20)
resp.raise_for_status()
all_poems_raw = resp.json()
if not isinstance(all_poems_raw, list):
    raise RuntimeError(f'PoetryDB returned an unexpected response: {all_poem
POEMS = [p for p in all_poems_raw if len(p['lines']) >= 12]
if not POEMS:
    raise RuntimeError('PoetryDB returned no poems with at least 12 lines.')
print(f'Found {len(POEMS)} poems with ≥ 12 lines.')

# Show length distribution
linecounts = [len(p['lines']) for p in POEMS]
fig, ax = plt.subplots(figsize=(8, 3))
ax.hist(linecounts, bins=range(12, max(linecounts)+2), color='steelblue', ed
ax.set_xlabel('Line count'); ax.set_ylabel('# poems')
ax.set_title('Distribution of Poem Lengths (filtered ≥ 12 lines)')
plt.tight_layout(); plt.show()
```

```
Fetching poems from poetrydb.org...
Found 164 poems with ≥ 12 lines.
```



In [7]:
```
# ── Interactive poem viewer ────────────────────────────────────────
POEM_TITLES = [f"{p['title']} — {p['author']}" for p in POEMS]

poem_dropdown = widgets.Dropdown(
    options=[(title, i) for i, title in enumerate(POEM_TITLES)],
    description='Poem:',
    layout=widgets.Layout(width='70%'),
)
ctx_slider = widgets.IntSlider(
    value=10, min=8, max=15, step=1,
    description='Context lines:',
    style={'description_width': 'initial'},
)
out_poem = widgets.HTML()

def show_poem(change=None):
    idx = poem_dropdown.value
    poem = POEMS[idx]
    lines = poem['lines']
    ctx_slider.max = min(15, len(lines) - 4)
```

```
    n_ctx = ctx_slider.value
    out_poem.value = f"""
    <div style='font-family:Georgia,serif; padding:12px; border:1px solid #d
      <h3 style='margin-bottom:4px'>{escape(poem['title'])}</h3>
      <em style='color:#666'>— {escape(poem['author'])}</em>
      <hr/>
      <div style='margin-top:8px; line-height:1.8'>
        {''.join(f"<span style='color:#1a5276; font-weight:bold'>{escape(l)}
        {''.join(f"<span style='color:#aaa'>{escape(l)}</span><br/>" for l i
      </div>
      <div style='margin-top:8px; font-size:0.85em; color:#555'>
        <b>Blue</b> = prompt context ({n_ctx} lines)  | 
        <b style='color:#aaa'>Grey</b> = held-out ground truth
      </div>
    </div>"""

poem_dropdown.observe(show_poem, names='value')
ctx_slider.observe(show_poem, names='value')
show_poem()
display(widgets.VBox([poem_dropdown, ctx_slider, out_poem]))
```

```
VBox(children=(Dropdown(description='Poem:', layout=Layout(width='70%'), opt
ions=(('Song of the Universal — Wa…
```

Section 3 — Reward Function Demo

In [8]:

```
# ── Live reward scorer ──────────────────────────────────────────────
text_input = widgets.Textarea(
    value='The sun rose bright over golden hills,\nAnd all the world was fil
    description='Text:',
    layout=widgets.Layout(width='70%', height='120px'),
)
score_btn = widgets.Button(description='Score it', button_style='primary')
out_score = widgets.HTML()

def on_score(b):
    result = score_text(text_input.value)
    bar_width = int(abs(result['sentiment']) * 300)
    color = '#27ae60' if result['sentiment'] > 0 else '#e74c3c'
    out_score.value = f"""
    <div style='font-family:monospace; padding:12px; background:#f8f9fa; bor
      <b>Label:</b> {result['label']}<br/>
      <b>Sentiment score:</b> {result['sentiment']:+.4f}  
        <span style='display:inline-block; width:{bar_width}px; height:12px;
                     background:{color}; border-radius:3px; vertical-align:m
      <b>Line count bonus:</b> {result['line_count_bonus']}  
        <span style='color:#888'>(newlines detected: {result['n_newlines']})
      <b>Total reward:</b> <span style='font-size:1.2em; color:{color}'>{res
    </div>"""

score_btn.on_click(on_score)
on_score(None)
display(widgets.VBox([text_input, score_btn, out_score]))
```

```
VBox(children=(Textarea(value='The sun rose bright over golden hills,\nAnd a
ll the world was filled with light…
```

In [9]:
```
# ── Batch score some hand-picked examples ──────────────────────────
examples = [
    ('Uplifting', 'The light broke through the morning grey,\nAnd filled the
    ('Neutral',   'The clock ticked slow upon the shelf,\nThe room was still
    ('Dark',      'The grave was cold, the wind was thin,\nNo star relieved
]
print(f'{"Example":<12}  {"Sentiment":>10}  {"Bonus":>6}  {"Total Reward":>1
print('-' * 50)
for label, text in examples:
    r = score_text(text)
    print(f'{label:<12}  {r["sentiment"]:>+10.4f}  {r["line_count_bonus"]:>6
```

```
Example        Sentiment   Bonus   Total Reward
--------------------------------------------------
Uplifting        +0.9999     1.0        +1.1999
Neutral          -0.9985     1.0        -0.7985
Dark             -0.9985     1.0        -0.7985
```

Section 4 — 🔴 Live GRPO Training

We run **two sequential training runs** with different β values:

- **β = 0.04** — aggressive; model drifts far from base
- **β = 0.4** — conservative; model stays close to base

Each training cell shows its own live curves after step 1 and every 5 steps thereafter.

In [10]:
```
def load_trainer_log(checkpoint_dir):
    with (checkpoint_dir / 'trainer_state.json').open() as file:
        return pd.DataFrame(json.load(file)['log_history'])

def plot_curves(df_low, df_high, ax_reward, ax_kl):
    for df, label, color in [(df_low, 'β=0.04', '#e74c3c'), (df_high, 'β=0.4
        reward_col = next(
            (col for col in ('reward', 'rewards/mean', 'rewards/reward_fn/me
            None,
        )
        for ax, metric in [(ax_reward, reward_col), (ax_kl, 'kl')]:
            if metric in df and 'step' in df:
                points = df.dropna(subset=['step', metric])
                ax.plot(points['step'], points[metric], color=color, label=l
                        linewidth=2, marker='.')

    ax_reward.set(title='Mean Reward vs Step', ylabel='Reward', xlabel='Step
    ax_kl.set(title='KL Divergence vs Step', ylabel='KL', xlabel='Step')
    for ax in (ax_reward, ax_kl):
        if ax.lines:
            ax.legend()
        ax.grid(alpha=0.3)
```

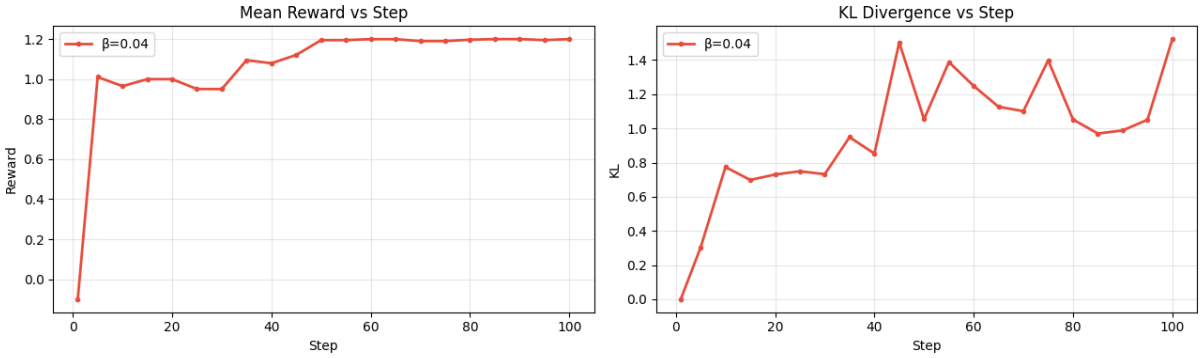
```
def render_curves(df_low, df_high):
    from io import BytesIO
    from matplotlib.backends.backend_agg import FigureCanvasAgg
    from matplotlib.figure import Figure

    fig = Figure(figsize=(13, 4), layout='tight')
    FigureCanvasAgg(fig)
    plot_curves(df_low, df_high, *fig.subplots(1, 2))
    with BytesIO() as image:
        fig.savefig(image, format='png', dpi=100)
        return Image(data=image.getvalue(), format='png')


def run_training(beta, checkpoint, prior_log=None):
    command = [
        sys.executable, '-u', str(PROJECT_ROOT / 'training' / 'train_grpo.py
        '--model', BASE_MODEL_ID, '--beta', str(beta), '--max_poems', '400',
        '--logging_steps', '5', '--max_steps', str(TRAIN_MAX_STEPS),
        '--output_dir', str(checkpoint),
    ]
    metrics = []
    recent_log = deque(maxlen=12)
    print(f'Launching β={beta}. The graph appears after the first training s

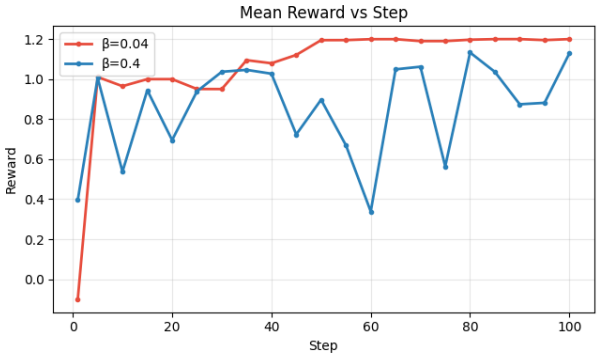
    with subprocess.Popen(
        command, cwd=PROJECT_ROOT, stdout=subprocess.PIPE, stderr=subprocess
        text=True, encoding='utf-8', errors='replace',
    ) as proc:
        try:
            for line in proc.stdout:
                # tqdm may leave a progress update on the same line.
                progress, marker, payload = line.partition('GRPO_METRICS ')
                if progress.strip():
                    recent_log.append(progress.rstrip())
                if marker:
                    metrics.append(json.loads(payload))
                    current = pd.DataFrame(metrics)
                    if prior_log is None:
                        chart = render_curves(current, pd.DataFrame())
                    else:
                        chart = render_curves(prior_log, current)
                    clear_output(wait=True)
                    display(chart)
                    print('\n'.join(recent_log), flush=True)
                elif progress.strip():
                    print(progress.rstrip(), flush=True)
            status = proc.wait()
        except BaseException:
            if proc.poll() is None:
                proc.terminate()
            raise
    if status:
        raise RuntimeError(f'β={beta} training exited with code {status}. Se
    return load_trainer_log(checkpoint)
```

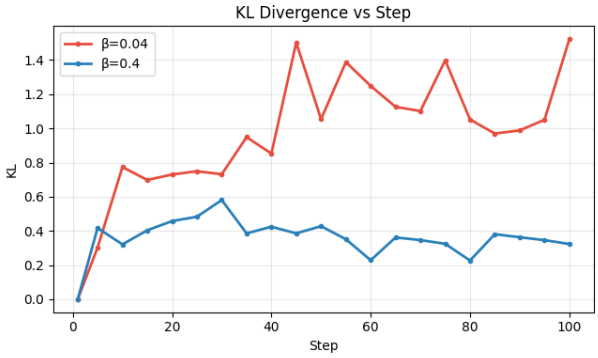
In [11]:
```
df_low = run_training(0.04, CHECKPOINT_LOW)
```



```
 91%|█████████ | 91/100 [04:35<00:21,  2.35s/it]
 92%|█████████ | 92/100 [04:37<00:18,  2.35s/it]
 93%|█████████ | 93/100 [04:40<00:17,  2.47s/it]
 94%|█████████ | 94/100 [04:42<00:14,  2.36s/it]
 95%|█████████ | 95/100 [04:44<00:11,  2.37s/it]
{'loss': '0.0121', 'grad_norm': '10.47', 'learning_rate': '6e-07', 'num_toke
ns': '1.693e+05', 'completions/mean_length': '45.12', 'completions/min_lengt
h': '37.8', 'completions/max_length': '55.2', 'completions/clipped_ratio': '
0', 'completions/mean_terminated_length': '45.12', 'completions/min_terminat
ed_length': '37.8', 'completions/max_terminated_length': '55.2', 'rewards/re
ward_fn/mean': '1.195', 'rewards/reward_fn/std': '0.01437', 'reward': '1.195
', 'reward_std': '0.01437', 'frac_reward_zero_std': '0', 'kl': '1.051', 'ent
ropy': '1.774', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0', 'cl
ip_ratio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_ma
x': '0', 'step_time': '2.401', 'epoch': '0.3084'}
 95%|█████████ | 95/100 [04:44<00:11,  2.37s/it]
 96%|█████████ | 96/100 [04:47<00:09,  2.37s/it]
 97%|█████████ | 97/100 [04:49<00:06,  2.31s/it]
 98%|█████████ | 98/100 [04:51<00:04,  2.27s/it]
 99%|█████████ | 99/100 [04:53<00:02,  2.25s/it]
100%|██████████| 100/100 [04:55<00:00,  2.25s/it]
{'loss': '0.06025', 'grad_norm': '10.24', 'learning_rate': '1e-07', 'num_tok
ens': '1.767e+05', 'completions/mean_length': '42.8', 'completions/min_lengt
h': '37.6', 'completions/max_length': '47.4', 'completions/clipped_ratio': '
0', 'completions/mean_terminated_length': '42.8', 'completions/min_terminate
d_length': '37.6', 'completions/max_terminated_length': '47.4', 'rewards/rew
ard_fn/mean': '1.2', 'rewards/reward_fn/std': '0.0001004', 'reward': '1.2',
'reward_std': '0.0001004', 'frac_reward_zero_std': '0', 'kl': '1.523', 'entr
opy': '1.764', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0', 'cli
p_ratio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_max':
'0', 'step_time': '2.163', 'epoch': '0.3247'}
100%|██████████| 100/100 [04:55<00:00,  2.25s/it]
{'loss': '0.06025', 'grad_norm': '10.24', 'learning_rate': '1e-07', 'num_tok
ens': '1.767e+05', 'completions/mean_length': '42.8', 'completions/min_lengt
h': '37.6', 'completions/max_length': '47.4', 'completions/clipped_ratio': '
0', 'completions/mean_terminated_length': '42.8', 'completions/min_terminate
d_length': '37.6', 'completions/max_terminated_length': '47.4', 'rewards/rew
ard_fn/mean': '1.2', 'rewards/reward_fn/std': '0.0001004', 'reward': '1.2',
'reward_std': '0.0001004', 'frac_reward_zero_std': '0', 'kl': '1.523', 'entr
opy': '1.764', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0', 'cli
p_ratio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_max':
'0', 'step_time': '2.163', 'epoch': '0.3247'}
100%|██████████| 100/100 [04:55<00:00,  2.25s/it]
Writing model shards:   0%|          | 0/1 [00:00<?, ?it/s]
Writing model shards: 100%|██████████| 1/1 [00:06<00:00,  6.60s/it]
Writing model shards: 100%|██████████| 1/1 [00:06<00:00,  6.60s/it]
{'train_runtime': '315.9', 'train_samples_per_second': '2.533', 'train_steps
_per_second': '0.317', 'train_loss': '0.0622', 'epoch': '0.3247'}
100%|██████████| 100/100 [05:15<00:00,  2.25s/it]
100%|██████████| 100/100 [05:15<00:00,  3.16s/it]
Writing model shards:   0%|          | 0/1 [00:00<?, ?it/s]
Writing model shards: 100%|██████████| 1/1 [00:19<00:00, 19.47s/it]
Writing model shards: 100%|██████████| 1/1 [00:19<00:00, 19.47s/it]
Done. Model saved to: /home/canders1/rlhf-s2/checkpoints/beta_0_04
```

In [12]:
```
df_high = run_training(0.4, CHECKPOINT_HIGH, prior_log=df_low)
```


Mean Reward vs Step
β=0.04
β=0.4
Reward
Step
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
20
40
60
80
100


KL Divergence vs Step
β=0.04
β=0.4
KL
Step
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
20
40
60
80
100

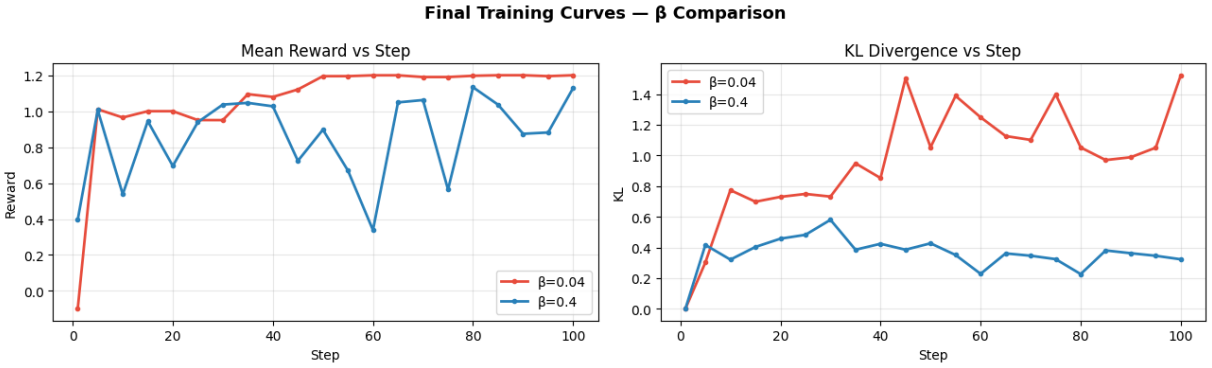
```
 91%|█████████ | 91/100 [04:15<00:24,  2.68s/it]
 92%|█████████ | 92/100 [04:17<00:20,  2.54s/it]
 93%|█████████ | 93/100 [04:19<00:16,  2.29s/it]
 94%|█████████ | 94/100 [04:21<00:14,  2.34s/it]
 95%|█████████ | 95/100 [04:24<00:12,  2.50s/it]
{'loss': '0.2298', 'grad_norm': '18.43', 'learning_rate': '6e-07', 'num_toke
ns': '1.603e+05', 'completions/mean_length': '55.48', 'completions/min_lengt
h': '37.2', 'completions/max_length': '84.4', 'completions/clipped_ratio': '
0.025', 'completions/mean_terminated_length': '53.55', 'completions/min_term
inated_length': '37.2', 'completions/max_terminated_length': '72.4', 'reward
s/reward_fn/mean': '0.8814', 'rewards/reward_fn/std': '0.683', 'reward': '0.
8814', 'reward_std': '0.683', 'frac_reward_zero_std': '0', 'kl': '0.346', 'e
ntropy': '2.444', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0',
'clip_ratio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_m
ax': '0', 'step_time': '2.467', 'epoch': '0.3105'}
 95%|█████████ | 95/100 [04:24<00:12,  2.50s/it]
 96%|█████████ | 96/100 [04:27<00:10,  2.53s/it]
 97%|█████████ | 97/100 [04:29<00:06,  2.33s/it]
 98%|█████████ | 98/100 [04:31<00:04,  2.38s/it]
 99%|█████████ | 99/100 [04:33<00:02,  2.21s/it]
100%|██████████| 100/100 [04:35<00:00,  2.09s/it]
{'loss': '0.2512', 'grad_norm': '18.31', 'learning_rate': '1e-07', 'num_toke
ns': '1.683e+05', 'completions/mean_length': '45.92', 'completions/min_lengt
h': '35', 'completions/max_length': '62.6', 'completions/clipped_ratio': '0'
, 'completions/mean_terminated_length': '45.92', 'completions/min_terminated
_length': '35', 'completions/max_terminated_length': '62.6', 'rewards/reward
_fn/mean': '1.129', 'rewards/reward_fn/std': '0.1738', 'reward': '1.129', 'r
eward_std': '0.1738', 'frac_reward_zero_std': '0', 'kl': '0.3231', 'entropy'
: '1.936', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0', 'clip_ra
tio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_max': '0'
, 'step_time': '2.048', 'epoch': '0.3268'}
100%|██████████| 100/100 [04:35<00:00,  2.09s/it]
{'loss': '0.2512', 'grad_norm': '18.31', 'learning_rate': '1e-07', 'num_toke
ns': '1.683e+05', 'completions/mean_length': '45.92', 'completions/min_lengt
h': '35', 'completions/max_length': '62.6', 'completions/clipped_ratio': '0'
, 'completions/mean_terminated_length': '45.92', 'completions/min_terminated
_length': '35', 'completions/max_terminated_length': '62.6', 'rewards/reward
_fn/mean': '1.129', 'rewards/reward_fn/std': '0.1738', 'reward': '1.129', 'r
eward_std': '0.1738', 'frac_reward_zero_std': '0', 'kl': '0.3231', 'entropy'
: '1.936', 'clip_ratio/low_mean': '0', 'clip_ratio/high_mean': '0', 'clip_ra
tio/region_mean': '0', 'clip_ratio/low_min': '0', 'clip_ratio/high_max': '0'
, 'step_time': '2.048', 'epoch': '0.3268'}
100%|██████████| 100/100 [04:35<00:00,  2.09s/it]
Writing model shards:   0%|          | 0/1 [00:00<?, ?it/s]
Writing model shards: 100%|██████████| 1/1 [00:05<00:00,  5.06s/it]
Writing model shards: 100%|██████████| 1/1 [00:05<00:00,  5.06s/it]
{'train_runtime': '293.6', 'train_samples_per_second': '2.725', 'train_steps
_per_second': '0.341', 'train_loss': '0.2305', 'epoch': '0.3268'}
100%|██████████| 100/100 [04:53<00:00,  2.09s/it]
100%|██████████| 100/100 [04:53<00:00,  2.94s/it]
Writing model shards:   0%|          | 0/1 [00:00<?, ?it/s]
Writing model shards: 100%|██████████| 1/1 [00:21<00:00, 21.27s/it]
Writing model shards: 100%|██████████| 1/1 [00:21<00:00, 21.27s/it]
Done. Model saved to: /home/canders1/rlhf-s2/checkpoints/beta_0_4
```

In [13]:
```
print('Loading both tuned models...')
```

```
tuned_model_low = AutoModelForCausalLM.from_pretrained(
    CHECKPOINT_LOW, dtype=MODEL_DTYPE, device_map='auto'
).eval()
tuned_model_high = AutoModelForCausalLM.from_pretrained(
    CHECKPOINT_HIGH, dtype=MODEL_DTYPE, device_map='auto'
).eval()

fig, (ax_reward, ax_kl) = plt.subplots(1, 2, figsize=(13, 4), layout='tight'
plot_curves(df_low, df_high, ax_reward, ax_kl)
fig.suptitle('Final Training Curves — β Comparison', fontsize=13, fontweight
plt.show()
```

```
Loading both tuned models...
Loading weights:   0%|          | 0/146 [00:00<?, ?it/s]
Loading weights:   0%|          | 0/146 [00:00<?, ?it/s]
```



Section 5 — Poem Sampler

Pick a poem, choose how many context lines to show, and compare completions from all three models.

In [14]:
```
def generate_completion(model, poem: dict, n_context: int, max_new_tokens: i
    messages = build_prompt(poem, n_context)
    inputs = tokenizer.apply_chat_template(
        messages, return_tensors='pt', return_dict=True, add_generation_prom
    ).to(model.device)
    with torch.inference_mode():
        output_ids = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            do_sample=True,
            temperature=0.8,
            top_p=0.95,
            pad_token_id=tokenizer.pad_token_id,
        )
    generated = output_ids[0, inputs['input_ids'].shape[1]:]
    return tokenizer.decode(generated, skip_special_tokens=True).strip()
```

In [15]:
```
# ── Sampler widget ──────────────────────────────────────────────────
sampler_dropdown = widgets.Dropdown(
    options=[(title, i) for i, title in enumerate(POEM_TITLES)],
```

```
    description='Poem:',
    layout=widgets.Layout(width='70%'),
)
sampler_slider = widgets.IntSlider(
    value=10, min=8, max=15, step=1,
    description='Context lines:',
    style={'description_width': 'initial'},
)
generate_btn = widgets.Button(
    description='⚡ Generate completions',
    button_style='success',
    layout=widgets.Layout(width='220px'),
)
out_sampler = widgets.HTML()

def card(title, text, reward_dict, color):
    label = reward_dict['label']
    total = reward_dict['total_reward']
    rcolor = '#27ae60' if total > 0 else '#e74c3c'
    lines_html = escape(text).replace('\n', '<br/>')
    return f"""
    <div style='flex:1; border:2px solid {color}; border-radius:8px; padding
                font-family:Georgia,serif; margin:4px; min-width:200px'>
      <div style='font-weight:bold; color:{color}; margin-bottom:8px; font-s
      <div style='line-height:1.8; color:#222'>{lines_html}</div>
      <hr style='margin:8px 0'/>
      <div style='font-family:monospace; font-size:0.85em; color:{rcolor}'>
        {label} · reward = {total:+.3f}
      </div>
    </div>"""

def on_generate(b):
    idx = sampler_dropdown.value
    poem = POEMS[idx]
    n_ctx = sampler_slider.value
    context_text = build_prompt(poem, n_ctx)[-1]['content']

    out_sampler.value = "<b>Generating... (may take 10–20s)</b>"

    comp_base = generate_completion(base_model,       poem, n_ctx)
    comp_low  = generate_completion(tuned_model_low,  poem, n_ctx)
    comp_high = generate_completion(tuned_model_high, poem, n_ctx)

    r_base = score_text(comp_base)
    r_low  = score_text(comp_low)
    r_high = score_text(comp_high)

    out_sampler.value = f"""
    <div style='font-family:Georgia,serif; margin-bottom:12px'>
      <h3 style='margin:0'>{escape(poem['title'])} <span style='font-weight:
      <div style='margin-top:8px; padding:10px; background:#f4f6f9; border-r
                  line-height:1.8; color:#1a5276; font-size:0.95em'>
        {escape(context_text).replace(chr(10), '<br/>')}
      </div>
      <div style='text-align:center; color:#888; margin:6px 0'>↓ model compl
    </div>
```

```
    <div style='display:flex; gap:8px'>
      {card('Base Llama 3.2 1B',    comp_base, r_base, '#7f8c8d')}
      {card('Tuned  β = 0.04',      comp_low,  r_low,  '#e74c3c')}
      {card('Tuned  β = 0.4',       comp_high, r_high, '#2980b9')}
    </div>
    """

def update_sampler_slider(change=None):
    poem = POEMS[sampler_dropdown.value]
    sampler_slider.max = min(15, len(poem['lines']) - 4)

sampler_dropdown.observe(update_sampler_slider, names='value')
update_sampler_slider()
generate_btn.on_click(on_generate)
display(widgets.VBox([
    sampler_dropdown,
    sampler_slider,
    generate_btn,
    out_sampler,
]))
```

```
VBox(children=(Dropdown(description='Poem:', layout=Layout(width='70%'), opt
ions=(('Song of the Universal — Wa…
```

Section 6 — Quantitative Comparison

In [16]:

```
# ── Score 20 random poems with all three models ──────────────────────
N_EVAL = min(20, len(POEMS))
eval_poems = random.sample(POEMS, N_EVAL)
results = []

print(f'Running inference on {N_EVAL} poems with 3 models...')
for i, poem in enumerate(eval_poems):
    n_ctx = random.randint(8, min(15, len(poem['lines']) - 4))
    for model, label in [
        (base_model,       'Base'),
        (tuned_model_low,  'β=0.04'),
        (tuned_model_high, 'β=0.4'),
    ]:
        comp = generate_completion(model, poem, n_ctx)
        r = score_text(comp)
        results.append({'poem': poem['title'], 'model': label,
                        'sentiment': r['sentiment'], 'total_reward': r['tota
    if (i + 1) % 5 == 0:
        print(f'  {i+1}/{N_EVAL} poems done...')

df_eval = pd.DataFrame(results)
print('Done!')
```

```
Running inference on 20 poems with 3 models...
```

```
[transformers] Ignoring clean_up_tokenization_spaces=True for BPE tokenizer
TokenizersBackend. The clean_up_tokenization post-processing step is designe
d for WordPiece tokenizers and is destructive for BPE (it strips spaces befo
re punctuation). Set clean_up_tokenization_spaces=False to suppress this war
ning, or set clean_up_tokenization_spaces_for_bpe_even_though_it_will_corrup
t_output=True to force cleanup anyway.
[transformers] You seem to be using the pipelines sequentially on GPU. In or
der to maximize efficiency please use a dataset
  5/20 poems done...
  10/20 poems done...
  15/20 poems done...
  20/20 poems done...
Done!
```

In [17]:

```
# ── Summary bar chart ──────────────────────────────────────────────
means = df_eval.groupby('model')['total_reward'].mean().reindex(['Base', 'β=
sems  = df_eval.groupby('model')['total_reward'].sem().reindex(['Base', 'β=0

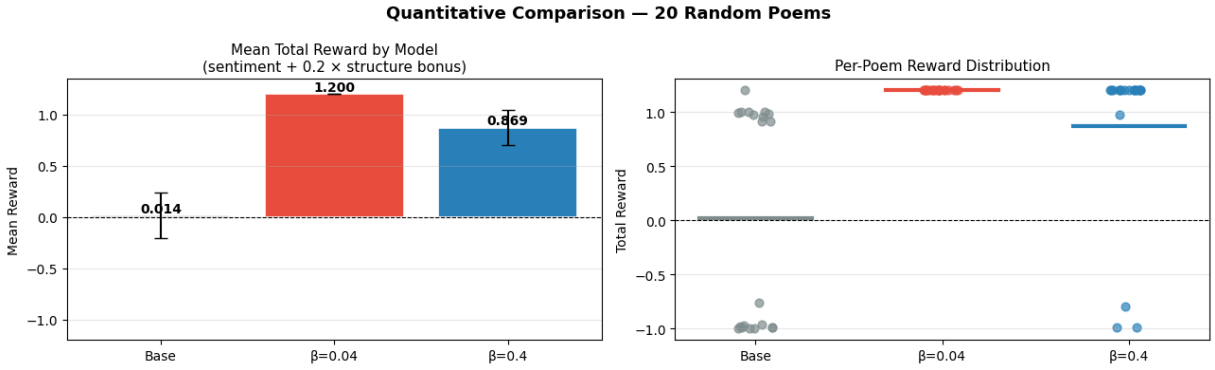
fig, axes = plt.subplots(1, 2, figsize=(13, 4))

# Bar chart: mean total reward
ax = axes[0]
colors = ['#7f8c8d', '#e74c3c', '#2980b9']
bars = ax.bar(means.index, means.values, yerr=sems.values, color=colors,
              capsize=5, edgecolor='white', linewidth=1.2)
ax.set_title('Mean Total Reward by Model\n(sentiment + 0.2 × structure bonus
ax.set_ylabel('Mean Reward'); ax.set_ylim(-1.2, 1.35)
ax.axhline(0, color='black', linewidth=0.8, linestyle='--')
ax.grid(axis='y', alpha=0.3)
for bar, v in zip(bars, means.values):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.03,
            f'{v:.3f}', ha='center', fontsize=10, fontweight='bold')

# Strip plot: individual poem scores
ax2 = axes[1]
for j, (label, color) in enumerate([('Base', '#7f8c8d'), ('β=0.04', '#e74c3c
    vals = df_eval[df_eval['model'] == label]['total_reward'].values
    jitter = np.random.uniform(-0.1, 0.1, size=len(vals))
    ax2.scatter(np.full_like(vals, j) + jitter, vals, color=color, alpha=0.7
    ax2.plot([j - 0.3, j + 0.3], [vals.mean(), vals.mean()], color=color, li

ax2.set_xticks([0, 1, 2]); ax2.set_xticklabels(['Base', 'β=0.04', 'β=0.4'])
ax2.set_title('Per-Poem Reward Distribution', fontsize=11)
ax2.set_ylabel('Total Reward')
ax2.axhline(0, color='black', linewidth=0.8, linestyle='--')
ax2.grid(axis='y', alpha=0.3)

plt.suptitle(f'Quantitative Comparison — {N_EVAL} Random Poems', fontsize=13
plt.tight_layout()
plt.show()
```


Quantitative Comparison — 20 Random Poems
Mean Total Reward by Model
(sentiment + 0.2 × structure bonus)
Mean Reward
0.014
1.200
0.869
Base
β=0.04
β=0.4
Per-Poem Reward Distribution
Total Reward
Base
β=0.04
β=0.4