3. Mid-Training

From CPT to Agentic RL: Foundations, recipes, and failure modes

Table of contents

0.1 Learning goals

Beyond memorizing definitions, interviews test whether you can **choose the right stage** (CPT vs SFT vs RL vs distill) and **anticipate failure modes**.

Tip

ELI5: *Training an LLM is like training a new hire: first you teach broad skills, then company-specific knowledge, then "how we talk to customers," and finally you coach them with feedback on real tasks.*

By the end of this chapter, you should be able to:

- Explain *why* mid-training (CPT) exists and how it differs from pretraining and SFT.
- Implement critical data strategies: **General Replay** for memory, **packing** for throughput, and **chat templates + masking** for instruction tuning.
- Apply **PEFT** (LoRA/QLoRA) strategies, including **multi-tenant serving** patterns.
- Compare alignment techniques: **PPO/RLHF** vs. **DPO/ORPO** vs. **Agentic RL** (e.g., **GRPO** and related variants).
- Design pipelines that optimize for **Reasoning (System 2)** using verifiers and process rewards.
- Build reliable **tool use** systems (schema correctness + tool selection + chaining).
- Apply **test-time scaling** (sampling, revision, verifier reranking) to boost reliability.
- Debug common regressions like **the stability gap**, **reward hacking**, and **template mismatch**.

0.2 The big picture: The life cycle of an LLM

This chapter organizes the landscape into a decision flow: *What's missing: knowledge, behavior, preferences, reasoning reliability, or cost?* Each corresponds to a different lever.

0.2.1 Interview framing: "Which knob would you turn?"

When you're given a product requirement, answer in this order:

1. **Define the target behavior** (format, safety, tool use, reasoning, latency).
2. **Diagnose the gap** (knowledge gap vs behavior gap vs optimization gap).
3. **Pick the cheapest effective lever** (prompt → SFT → DPO → RL → CPT → distill).
4. **Name the regression risks** (forgetting, reward hacking, template mismatch, latency).

> Note
>
> **ELI5:** *If the model doesn't know the facts, teach it with reading (CPT). If it knows facts but talks wrong, teach it with examples (SFT). If it needs to prefer "better" answers, use preferences/RL. If it's too slow/expensive, compress it.*

We treat training and inference as a pipeline of altering distributions *and* compute budgets.

```
flowchart LR
  A[Base Model  0] --> B{Domain gap?}

  B -->|Yes: Knowledge/Vocab/Context| C[Mid-training / CPT]
  B -->|No: Just behavior| D[SFT]

  C --> E[ _mid]
  E --> D
  D --> F[ _sft]

  F --> G{Alignment path}
  G -->|Chat/Style| H[DPO / ORPO / PPO]
  G -->|Reasoning/Math/Code| I[Agentic RL: GRPO / Self-training loops]

  H --> J[ _aligned]
  I --> J

  J --> K{Inference budget?}
  K -->|Low| L[Direct decode]
  K -->|High| M[Test-time scaling]

  J --> N[Distillation / Quantization]
  N --> O[ _deploy]
```

Note

Core mental model

- **Pretraining:** builds the engine (general capabilities).
- **Mid-training (CPT):** tunes the engine for terrain (domain knowledge, context-length priors, sometimes RL-compatibility).
- **Post-training (SFT/DPO/RL):** teaches the driver (behavior, safety, preference alignment).
- **Test-time scaling:** spends compute *at inference* to navigate complex routes (better reliability without retraining).

0.3 Data topology: the hidden interview topic

Most real failures come from feeding the right data through the **wrong loss topology**.

0.3.1 The three "topologies" you should be able to explain

- **Packing (CPT):** maximize tokens/GPU by concatenating documents to fill the context window.
- **Masking (SFT):** compute loss only where you want supervision (typically assistant tokens).
- **Rollouts (RL):** the policy generates full sequences; you score outcomes and optimize expected reward.

0.3.2 Quick mental check

Ask: *"Which tokens contribute gradients?"*
- CPT: **all tokens** (next-token loss)
- SFT: **assistant tokens only** (masked)
- DPO: **chosen vs rejected** (contrast)
- RL: **tokens sampled by the policy** (on-policy)

Tip

ELI5: *Same text, different learning: CPT learns to continue text, SFT learns to answer like a tutor, RL learns by trying things and getting a score.*

A common interview failure is not distinguishing how data is *constructed* and how loss is *applied*.

Tip

Packing vs masking vs rollouts
- **Packing:** concatenating docs to fill context (often used in pretraining/CPT; sometimes also used in SFT for throughput).
- **Masking:** loss only on assistant tokens (typical in SFT on chat transcripts).
- **Rollouts:** generating full sequences until EOS (RL/agentic RL). Enables exploration and verifier-based selection.

```
flowchart LR
  A[Raw text / docs] --> B[CPT: next-token loss (often packed)]
  C[Instruction chats] --> D[SFT: masked loss on assistant tokens]
  E[Preference pairs] --> F[DPO/ORPO: contrast chosen vs rejected]
  G[Prompts + Verifier/Env] --> H[Agentic RL: rollouts + scoring + optimize]
```

1 Phase 1: Mid-training / Continued Pretraining (CPT)

CPT is the workhorse for **domain specialization** and **long-context priors**. It's also the most common place to introduce regressions if you don't guardrail general capabilities.

Tip

ELI5: *CPT is "more reading": you keep training the model on domain documents so it picks up jargon and facts naturally.*

1.0.1 When CPT is the right tool

Use CPT when you see: - high perplexity on in-domain corpora, - systematic entity/jargon failures, - domain-specific formatting/structures (legal docs, codebases), - long-document understanding gaps.

1.0.2 CPT design checklist (interview-ready)

- **Data:** quality > quantity; de-dup, remove boilerplate, enforce doc boundaries.
- **Mixture:** start with replay (e.g., 80/20) and tune by regression gates.
- **LR:** typically much lower than initial pretraining peak LR.
- **Eval:** track both *domain gains* and *general regressions* continuously.

1.1 What it is (and isn't)

1.1.1 What CPT optimizes

CPT uses the **same next-token objective** as base pretraining, but on a **different distribution**.

- It tends to improve *knowledge recall* and *domain fluency*.
- It can also improve *tool-use* and *reasoning* indirectly when your domain data contains those patterns (e.g., code repos, math solutions).

1.1.2 What CPT is not

- Not instruction-following by itself: you can CPT a model into a great encyclopedia that still refuses to answer in JSON.
- Not a replacement for alignment: CPT can even make safety worse if your corpus contains unsafe patterns.

Note

ELI5: *CPT teaches what to say; SFT teaches how to say it to a user.*

Continued **next-token training** on a target distribution. It is necessary when the base model lacks:

- **domain knowledge** (facts/jargon/entity priors),
- **long-context priors** (document structure, long-range retrieval),
- and sometimes **RL-compatibility** for reasoning-style RL (model-family differences).

It is *not* SFT: SFT is about behavioral formatting and instruction following.

1.2 Optional: Tokenizer extension

Tokenizer extension can be worth it when your domain has many **high-frequency terms** that get split into many sub-tokens (e.g., rare drug names, API identifiers, legal citations).

Tip

ELI5: *Tokenizer extension is like adding new dictionary words so the model stops spelling domain terms letter-by-letter.*

1.2.1 When to extend

- **Decision rule:** measure **fragmentation rate** on a domain vocabulary list. If important terms routinely become 5+ tokens, you're wasting context and compute.
- **Signal:** you see frequent truncation/length issues, or the model mangles domain terms (broken identifiers, misspellings, bad citations).

1.2.2 How to extend (minimal-risk workflow)

1. Compile a vocabulary shortlist (top entities, terms, APIs).
2. Add tokens for the worst offenders.
3. Resize embeddings; initialize new token rows (random, or average of constituent sub-token embeddings).
4. Warm up: oversample examples containing the new tokens for the first phase of CPT.

1.2.3 Failure modes to mention

- **Undertrained tokens:** new tokens behave like noise early → mitigated via oversampling and warm-up.

- **Segmentation mismatch:** any downstream pipeline that tokenizes text must use the updated tokenizer.
- **Distribution shock:** adding tokens changes token counts and packing; re-check sequence length assumptions.

1.3 The stability gap

1.3.1 Why it happens

Early in CPT, gradients strongly adapt the model to the new distribution, often pushing it away from general-purpose representations.

Common contributors: - too-high learning rate, - low diversity domain data (narrow corpora), - insufficient general replay, - noisy/mislabeled documents (scrapes, templated boilerplate).

1.3.2 Debugging playbook

If general benchmarks drop sharply: - decrease LR and/or increase warm-up, - increase replay ratio, - improve data quality (dedup, filter spam), - add **regularization to the reference** (e.g., KL to base logits if available).

Note

ELI5: *Stability gap is like cramming for one exam and temporarily forgetting other subjects—replay is doing a little "general homework" to prevent that.*

CPT often causes an early dip in general performance before recovering.

- **Mitigation: General replay** (e.g., 80% domain / 20% replay as a starting point), plus regression gates.

```
# Pseudocode: CPT loop with packing + replay

### Practical add-ons (what interviews like)
- **Curriculum:** start with higher replay, then anneal toward more domain.
- **Domain mixing:** if multiple sub-domains, use adaptive sampling (upweight domains with h
- **Guardrails:** run a small regression suite every N steps; stop/rollback on big drops.

for step in range(T):
    batch = sample(D_domain) if rand() < p_domain else sample(D_replay)

    # Packing: concatenate docs to fill context (no padding; delimit with EOS)
    x = pack_sequences(batch, seq_len=L)

    # Next-token objective
    loss = cross_entropy(model(x[:, :-1]), x[:, 1:])
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    if step % eval_every == 0 and regression_failed():
        tune(p_domain="down", lr="down", data_quality="up")
```

Note

Mid-training to enable RL scaling (optional add-on)
Two useful framing papers for the “CPT → RL compatibility” story are:

- *OctoThinker: Mid-training Incentivizes Reinforcement Learning Scaling* (arXiv:2506.20512) [@octothinker_2025]

- *On the Interplay of Pre-Training, Mid-Training, and RL on Reasoning Language Models* (arXiv:2512.07783) [@interplay_pre_mid_rl_2025]