discuss supervised fine tuning and compare and contrast it to reinforcement learning with verified rewards.

These are two important techniques for improving language models after pretraining, each with distinct mechanisms and trade-offs.

Supervised Fine-Tuning (SFT)

SFT involves training a pretrained model on curated input-output pairs. You provide examples of desired behavior—prompts paired with ideal responses—and the model learns to imitate them by minimizing the cross-entropy loss between its predictions and the target tokens.

The process is straightforward: collect high-quality demonstrations (often from human experts), format them appropriately, and train. SFT is effective for teaching a model a particular style, format, or domain-specific knowledge. It's also relatively stable and well-understood from a training dynamics perspective.

However, SFT has limitations. The model is bounded by the quality of the demonstrations—it learns to imitate but doesn't necessarily learn to exceed its teachers. It also treats all tokens equally, which can be inefficient when some parts of a response matter more than others. And because it only sees "correct" examples, it doesn't learn from mistakes or develop robust strategies for avoiding failure modes.

Reinforcement Learning with Verified Rewards (RLVR)

RLVR is a form of reinforcement learning where the reward signal comes from an automated verifier rather than a learned reward model or human preferences. This is particularly powerful in domains where correctness can be checked programmatically—mathematics (does the answer match?), code (does it pass tests?), or formal reasoning tasks.

The model generates candidate responses, the verifier scores them (often binary: correct or incorrect), and the model updates its policy to increase the probability of responses that earned positive rewards. Algorithms like PPO, REINFORCE, or more recent approaches like GRPO are commonly used.

The key advantage is that RLVR lets the model explore and discover strategies that may not exist in any training demonstration. A model can stumble onto a novel proof technique or coding approach, receive a verified reward, and reinforce that behavior—even if no human ever demonstrated it. This enables genuine capability improvement beyond imitation.

Key Differences

The learning signal differs fundamentally. SFT uses dense, token-level supervision from demonstrations. RLVR uses sparse, outcome-level feedback from verification. This makes RLVR more sample-inefficient but potentially more powerful for capability elicitation.

Generalization patterns diverge as well. SFT models interpolate within the distribution of demonstrations. RLVR models can extrapolate by exploring the space of possible solutions and receiving feedback on what works. This is why RLVR has been particularly successful for improving mathematical reasoning—the model can discover solution paths not present in training data.

Training stability is another distinction. SFT is generally stable and predictable. RLVR requires careful tuning of reward scaling, KL penalties (to prevent the model from drifting too far from a reference policy), and other hyperparameters. Reward hacking—where models find unintended ways to achieve high reward—is a persistent concern.

The scope of applicability also differs. SFT works for any task where you can provide demonstrations. RLVR requires a reliable verifier, which limits it to domains with objective correctness criteria. You can't easily verify whether a poem is beautiful or an explanation is clear, so those tasks remain better suited to SFT or RLHF with human preferences.

Complementary Roles

In practice, these methods are often combined. A common pipeline starts with SFT to establish baseline capabilities and response format, then applies RLVR (where verifiers exist) or RLHF to refine behavior further. The SFT stage gives the model enough competence that its RL explorations are productive rather than random, while the RL stage pushes capabilities beyond what demonstrations alone could achieve.

What does this paper teach us about the issue? __https://arxiv.org/pdf/2511.08567__

2 steps

arxiv 2511.08567 paper 10 results

This is a fascinating paper that offers deep mechanistic insights into the SFT vs. RLVR comparison. Here's what it teaches us:

The Central Paradox

RLVR reliably improves reasoning performance in large language models, yet it appears to modify only a small fraction of parameters. arXiv The authors investigated why this sparsity occurs and discovered it's actually a surface artifact of something deeper.

The Three-Gate Theory

The paper proposes a mechanistic explanation for RLVR's distinctive optimization dynamics:

Gate I (KL Anchor) imposes a KL-constrained update; Gate II (Model Geometry) steers the step off principal directions into low-curvature, spectrum-preserving subspaces; and Gate III (Precision) hides micro-updates in non-preferred regions, making the off-principal bias appear as sparsity. arXiv

This explains why RLVR updates look sparse—it's not that few parameters change, but that the optimization naturally gravitates toward specific regions of parameter space that preserve the model's existing structure.

The Fundamental Difference from SFT

Here's the key insight for our discussion: RLVR learns off principal directions in weight space, achieving gains via minimal spectral drift, reduced principal-subspace rotation, and off-principal update alignment. In contrast, SFT targets principal weights, distorts the spectrum, and even lags RLVR. arXiv

In simpler terms, SFT modifies the "main" weight directions that define the model's core representations—which can distort what the model has already learned. RLVR instead finds secondary directions that add capability without disrupting existing structure. This is why RLVR can improve reasoning while preserving general capabilities.

Practical Implications

RL operates in a distinct optimization regime from SFT, so directly adapting SFT-era parameter-efficient fine-tuning (PEFT) methods can be flawed, as evidenced by case studies on advanced sparse fine-tuning and LoRA variants. arXiv

This has significant practical consequences: techniques like LoRA, which were designed for SFT's optimization landscape, may not transfer well to RLVR. The geometry of how each method navigates parameter space is fundamentally different, so PEFT methods need to be redesigned for RL rather than simply borrowed.

Why This Matters for the Broader Comparison

This paper deepens our understanding of why RLVR can push capabilities beyond what SFT achieves. It's not just that RL explores and gets feedback—it's that the optimization itself respects the model's geometry in a way that preserves learned structure while adding new capabilities in orthogonal directions. SFT's imitation learning, by contrast, overwrites principal components, which may explain why it can sometimes degrade capabilities even while teaching new behaviors.

Discuss the origin of the KL leash. How does it emerge in RLVR?

The KL divergence constraint—often called the "KL leash," "KL anchor," or "KL penalty"—is a regularization mechanism that prevents the policy (the model being trained) from drifting too far from a reference policy (typically the pretrained or SFT-initialized model). Understanding its origin requires tracing through both the theoretical motivations and practical necessities.

Theoretical Origins

The KL constraint emerges from the framework of **trust region optimization**. The core idea dates back to conservative policy iteration and was formalized for deep RL in TRPO (Trust Region Policy Optimization) by Schulman et al. (2015).

The intuition is straightforward: when you update a policy based on reward signals, you're making decisions based on data collected under the *old* policy. If your new policy diverges too much, the data you used to estimate gradients becomes unreliable—you're extrapolating far beyond where you have evidence. The KL constraint keeps updates within a "trust region" where your gradient estimates remain valid.

For language models specifically, this constraint takes the form:

$$\mathrm{L} = \mathrm{E[reward]} - \beta \cdot D_{KL}(\pi_\theta \| \pi_{\text{ref}})$$

where π_θ is the current policy, π_{ref} is the reference model, and β controls how tightly the leash is held.

Why It's Essential in RLVR

In RLVR for language models, the KL penalty serves several interconnected purposes:

Preventing reward hacking. Verifiable rewards, while objective, are often narrow. A math verifier checks if the final answer is correct but doesn't evaluate the reasoning path's elegance or the response's coherence. Without a KL constraint, the model could find degenerate solutions—garbled text that somehow produces correct answers, or exploiting quirks in the verification system. The KL penalty forces the model to stay close to "normal" language model behavior while optimizing for correctness.

Preserving general capabilities. The pretrained model embodies vast knowledge and capabilities beyond the specific task being optimized. Unconstrained optimization would catastrophically forget these. The KL leash acts as a soft constraint saying "improve on this task, but don't become a fundamentally different model."

Stabilizing training. Language model outputs are high-dimensional probability distributions over vocabulary. Small parameter changes can cause large distributional shifts in this space. The KL penalty smooths the optimization landscape, preventing the wild oscillations that would otherwise occur.

How It Manifests in the Three-Gate Theory

Returning to the paper you shared, the KL constraint is Gate I in their framework. But their contribution is showing that this isn't just a regularizer slapped onto the objective—it fundamentally shapes *where* in parameter space learning occurs.

The KL penalty creates a soft boundary around the reference model. Updates that would push the output distribution far from the reference incur heavy penalties. This constraint interacts with the model's geometry (Gate II) to channel updates into specific subspaces—namely, the "off-principal" directions that preserve the model's spectral structure.

Think of it this way: the principal directions in weight space are the ones most responsible for the model's current output distribution. Modifying them would immediately violate the KL constraint. So the optimization naturally finds the secondary directions where it can make progress on the reward objective without triggering the KL penalty.

Implementation Variations

The KL leash appears in different algorithmic forms:

PPO-style: The KL penalty is added directly to the reward, effectively treating deviation from the reference as a cost. The coefficient β may be fixed or adaptively adjusted to target a specific KL budget.

GRPO and variants: Group Relative Policy Optimization and similar methods often implicitly incorporate KL constraints through their update rules, comparing candidate completions relative to a baseline that's anchored to the reference distribution.

Explicit KL targeting: Some implementations dynamically adjust β to maintain the KL divergence within a target range, tightening the leash if the model drifts too far and loosening it if there's room to explore.

The Deeper Question

What the paper reveals is that the KL leash isn't just preventing bad outcomes—it's actively shaping the nature of the learning itself. By constraining *what* can change, it determines *how* improvement happens. RLVR doesn't just add reasoning capability on top of the base model; it finds a geometrically distinct path through parameter space that the KL constraint makes accessible while blocking the more destructive routes that SFT takes directly.

This is why simply removing or weakening the KL penalty doesn't lead to better RLVR—it leads to collapse. The constraint isn't fighting against learning; it's channeling learning into the regime where RLVR's distinctive benefits emerge.

Is this applicable to human learning?

This is a genuinely interesting question, and I think the answer is "suggestively yes, but with important caveats." The dynamics uncovered in RLVR may reflect deeper principles about how learning systems—biological or artificial—acquire new capabilities without destroying existing ones.

The Catastrophic Forgetting Problem

Both artificial neural networks and biological brains face a fundamental tension: learning requires modifying representations, but modification risks overwriting what's already there. In artificial networks, this manifests as catastrophic forgetting—train on a new task and performance on old tasks collapses. Yet humans learn continuously throughout life without forgetting how to walk every time they learn a new fact.

The brain must have something functionally analogous to the KL leash. And intriguingly, there's evidence it does.

Synaptic Consolidation as a Biological KL Constraint

Neuroscience has identified mechanisms that protect important synaptic connections from modification. The theory of **synaptic consolidation** suggests that synapses encoding well-established memories become progressively more resistant to change through biochemical processes involving protein synthesis and structural modifications.

More formally, **metaplasticity**—the plasticity of plasticity itself—means that a synapse's susceptibility to change depends on its history. Heavily-used, task-critical synapses become "stabilized," while less-committed synapses remain available for new learning. This is strikingly analogous to how the KL constraint in RLVR protects principal directions while permitting updates in secondary subspaces.

The computational neuroscience framework of **Elastic Weight Consolidation** (developed by DeepMind, interestingly) was explicitly inspired by this biology. It penalizes changes to parameters proportionally to their importance for previous tasks—a direct analogue of the KL penalty.

Sleep and Off-Principal Consolidation

During sleep, the brain doesn't simply rest—it actively reorganizes and consolidates memories. The evidence suggests that new learning is initially encoded in the hippocampus (a "fast learning" system) and then gradually integrated into cortical networks (a "slow learning" system) during sleep.

This two-system architecture may serve a function similar to RLVR's off-principal learning. New information is first captured in a buffer that doesn't disrupt existing cortical representations. Then, during consolidation, it's integrated in ways that preserve the structure of existing knowledge—finding the "directions" in neural representation space where new information can be added without corrupting what's there.

The slow-wave oscillations during deep sleep may literally be coordinating which representations get modified and which remain protected.

Expertise and Structural Preservation

Consider how expertise develops. A chess grandmaster doesn't replace their knowledge of how pieces move—that remains stable, automatic, protected. Instead, expertise builds increasingly sophisticated pattern recognition and strategic reasoning *on top of* the foundational knowledge.

This looks like off-principal learning: the "principal directions" encoding basic chess mechanics remain fixed, while capability is added in orthogonal dimensions. The expert and the novice share the same foundations; the expert has elaborated in directions the novice hasn't explored.

Similarly, a skilled musician doesn't forget basic motor patterns when learning a new piece. The new learning integrates with, rather than overwrites, existing technique. If anything, the foundations become *more* stable with expertise—more resistant to modification—while flexibility is channeled into higher-level representations.

Imitation vs. Discovery in Human Learning

The SFT vs. RLVR distinction also has human analogues. Consider two ways of learning to solve a type of math problem:

SFT-like: You watch worked examples and learn to imitate the demonstrated procedure. This is efficient for acquiring the procedure but may not generalize well—you've learned *what* to do without deeply understanding *why*.

RLVR-like: You attempt problems yourself, get feedback on whether your answers are correct, and gradually develop your own understanding. This is slower but often produces more robust, transferable knowledge.

Educational research broadly supports this. **Productive failure** paradigms—where students struggle with problems before receiving instruction—often outperform direct instruction. The struggle isn't wasted effort; it's building representations that instruction alone doesn't create. The student who discovers a solution has learned something geometrically different from the student who copies one.

The Zone of Proximal Development as a Trust Region

Vygotsky's **zone of proximal development** (ZPD) describes the space between what a learner can do independently and what they can do with guidance. Effective teaching operates within this zone—not too easy (no learning), not too hard (no traction).

This is conceptually similar to a trust region. Learning that stays close to current capabilities can make reliable progress. Learning that attempts to leap too far from current understanding fails because there's no foundation to connect with—the "gradient estimates" from experience become unreliable when extrapolating beyond the known.

Good teachers intuitively implement something like adaptive KL targeting: they adjust the difficulty to keep learning within the productive zone, tightening the constraint when students struggle and loosening it when they're ready for more.

Important Disanalogies

I should be careful not to overclaim. The brain and transformer language models are very different systems:

Architecture: The brain has specialized regions, recurrent connectivity, neuromodulation, and neurogenesis. Transformers have uniform layers, feedforward processing, and fixed parameter counts.

Learning signals: Human learning involves dopaminergic reward prediction errors, but also social feedback, intrinsic curiosity, emotional valence, and embodied experience. RLVR uses binary verification signals.

Timescales: Biological learning operates across milliseconds (synaptic potentiation), hours (consolidation), and years (development). RLVR operates in discrete training steps.

So while the *functional* pressures are similar—preserve existing capabilities while acquiring new ones—the *implementations* are likely quite different.

The Deeper Principle

What might be universal is this: any learning system that must accumulate capabilities over time, rather than training from scratch for each new task, faces the geometry problem that the RLVR paper characterizes. There are directions in representation space that encode existing functionality and directions that are relatively "free" for new learning.

Successful continual learning—whether in brains or machines—may require mechanisms that identify and protect the former while channeling updates toward the latter. The KL constraint in RLVR and synaptic consolidation in brains may be convergent solutions to the same underlying problem.

This suggests a research direction: can insights from how the brain solves this problem inform better RLVR algorithms? And can the mathematical clarity of the Three-Gate Theory illuminate what the brain's consolidation mechanisms are actually computing?

With respect to the teachings in the above paper, what is the relevance of arXiv:2506.00772?

arxiv 2506.00772 10 results

Based on the search results, this paper—"LIFT the Veil for the Truth: Principal Weights Emerge after Rank Reduction for Reasoning-Focused Supervised Fine-Tuning"—provides a fascinating counterpoint to the RLVR paper. The connection is striking and somewhat paradoxical.

The LIFT Paper's Core Finding

The authors state that weights with the largest magnitude after low-rank approximation are critical weights for fine-tuning, which they call Principal Weights. Surprisingly, while magnitude-based sparse fine-tuning performs poorly as a baseline on LLM fine-tuning, it becomes highly effective after rank reduction. arXiv

The paper addresses a practical problem: full fine-tuning, while powerful, is computationally expensive and susceptible to overfitting and catastrophic forgetting, particularly when data is limited. arXiv Their solution is to identify which parameters matter most for SFT and update only those.

The Paradox When Juxtaposed with the RLVR Paper

Here's where things get interesting. Recall what the RLVR paper established:

- RLVR learns **off** principal directions, preserving spectral structure
- SFT targets **principal** weights, distorting the spectrum
- This geometric difference explains why RLVR can improve reasoning while preserving capabilities, while SFT risks catastrophic forgetting

Now the LIFT paper comes along and says: for SFT specifically, if you want efficiency, you should focus updates *precisely* on the principal weights (after rank reduction reveals them).

These aren't contradictory findings—they're complementary observations about fundamentally different optimization regimes.

Reconciling the Two Perspectives

The apparent tension resolves when you consider what each paper is optimizing for:

LIFT's perspective: Given that you're doing SFT (imitation learning from demonstrations), the principal weights are where the action is. The learning signal from demonstration data naturally wants to modify these weights. LIFT just identifies them more efficiently so you can do sparse updates rather than full fine-tuning.

The RLVR paper's perspective: The reason RLVR outperforms SFT isn't just about reward signals—it's that the KL constraint forces learning into a geometrically different regime. RLVR *can't* efficiently modify principal directions without violating the KL penalty, so it finds gains in off-principal subspaces.

The implication is profound: **SFT and RLVR aren't just using different learning signals—they're literally operating in orthogonal subspaces of parameter space.**

Why This Matters

This juxtaposition suggests several insights:

For understanding capability acquisition: SFT and RLVR may be building different *types* of capabilities. SFT modifies core representations (principal directions), which is powerful but risky. RLVR adds capabilities that are somehow "orthogonal" to existing structure—augmenting rather than replacing.

For training pipelines: The standard pipeline (pretrain → SFT → RLHF/RLVR) may be doing something quite specific geometrically. SFT shapes the principal directions to establish format, style, and basic task competence. Then RLVR refines capabilities in the off-principal complement without disturbing what SFT established. They're not redundant—they're operating in complementary subspaces.

For parameter-efficient methods: LIFT shows you can make SFT efficient by targeting principal weights. But as the RLVR paper warned, these SFT-era PEFT methods may fail for RL precisely because RL needs access to the off-principal subspace. You'd need geometrically different sparse methods for each regime.

For catastrophic forgetting: LIFT acknowledges that full fine-tuning is susceptible to overfitting and catastrophic forgetting, particularly when data is limited. arXiv The RLVR paper explains *why*: SFT's principal-direction updates distort the spectrum, corrupting existing capabilities. RLVR's off-principal updates avoid this by construction.

A Unified Picture

Together, these papers suggest a geometric theory of fine-tuning:

1. **Principal subspace**: Encodes the model's core learned representations. SFT naturally targets this space. Modifications here are powerful but destabilizing.
2. **Off-principal subspace**: Represents "slack" in the model's parameterization. RLVR (constrained by KL) operates here. Modifications preserve structure while adding capability.
3. **Rank reduction** (LIFT's trick): Helps identify the principal subspace more cleanly, making SFT more efficient by focusing updates where they matter.

4. **The KL constraint** (RLVR's key): Acts as a gate that blocks principal-direction updates, channeling learning into the off-principal complement.

The two papers are essentially characterizing different regions of the same geometric landscape, explaining why different fine-tuning approaches have the properties they do.

What does arXiv:2502.11711 add to this discussion?

10 steps

"Does RLVR Really Improve Reasoning" arxiv paper 2025 0 results

arxiv RLVR pass@k boundary reasoning base model already 10 results

I wasn't able to directly access arXiv:2502.11711, but my search uncovered a closely related body of work—most notably arXiv:2504.13837, "Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?" This paper

Sorry it is 2505.11711. So the question is what does 2505.11711 add to this discussion?

arxiv 2505.11711 10 results

This is an excellent paper to add to the discussion. "Reinforcement Learning Finetunes Small Subnetworks in Large Language Models" (arXiv:2505.11711) provides crucial empirical evidence that strongly corroborates and extends the theoretical picture we've been building.

The Core Finding

Reinforcement learning yields substantial improvements in large language models' downstream arXiv task performance and alignment with human values. Surprisingly, such large gains result from updating only a small subnetwork comprising just 5 percent to 30 percent of the parameters, with the rest effectively unchanged.

This phenomenon—which the authors call "parameter update sparsity induced by RL"—is remarkably robust: It is observed across all 7 widely used RL algorithms (e.g., PPO, GRPO, DPO) and all 10 LLMs from different families in our experiments. This sparsity is intrinsic and occurs without any explicit sparsity promoting regularizations or architectural constraints. arXiv

How This Connects to the Earlier Papers

This paper provides the missing empirical bridge between the theoretical frameworks we've discussed:

Connection to the Three-Gate Theory (2511.08567): The RLVR paper argued that RL learns "off-principal" directions—that the KL constraint and model geometry channel updates into specific subspaces. This new paper provides direct empirical confirmation: RL really does update only a small, localized portion of the network. The sparsity pattern is the observable signature of the geometric constraints the Three-Gate Theory describes.

Connection to LIFT (2506.00772): LIFT showed that for SFT, the *principal weights* (identified after rank reduction) are critical. This paper shows RL updates a different, sparse subset. Together, they suggest SFT and RL are literally modifying different parameters—not just different "directions" in an abstract sense, but actually different weights in the network.

Connection to "SFT Memorizes, RL Generalizes" (2501.17161): Why does RL generalize better? This paper offers a mechanistic explanation: by updating only 5-30% of parameters while leaving the rest unchanged, RL preserves the pretrained model's broad capabilities. The sparse update pattern is a form of implicit regularization that prevents the kind of representational distortion that causes SFT to overfit.

Key Mechanistic Insights

Several findings are particularly illuminating:

Finetuning the subnetwork alone recovers the test accuracy, and, remarkably, produces a model nearly identical to the one obtained via full finetuning. arXiv

This is striking—you can identify which 5-30% of parameters RL wants to update, freeze everything else, and get essentially the same result. The learning is genuinely localized.

The subnetworks from different random seeds, training data, and even RL algorithms show substantially greater overlap than expected by chance. arXiv

This is perhaps the most important finding for our discussion. The subnetwork isn't random—it's determined by the *model's geometry*, not by the specific RL algorithm or data. Different RL methods converge on modifying the same parameters. This strongly supports the Three-Gate Theory's claim that it's the pretrained model's structure that determines where learning can occur.

The Full-Rank Paradox

A subtle but important observation: The updates to almost all parameter matrices are nearly full-rank, suggesting RL updates a small subset of parameters that nevertheless span almost the full subspaces that the parameter matrices can represent. arXiv

This resolves an apparent tension. RL updates are *sparse* (few parameters change) but *not low-rank* (they span the full representational capacity of those parameters). This means RL isn't just making small, constrained adjustments—it's making rich, expressive changes, but only to a specific subset of the network.

This distinguishes RL from methods like LoRA, which impose low-rank constraints. RL naturally finds a sparse-but-full-rank update pattern that LoRA's architecture cannot capture.

What Causes the Sparsity?

The authors investigate the mechanism: We conjecture that the this update sparsity can be primarily attributed to training on data that is near the policy distribution, techniques that encourage the policy to remain close to the pretrained model, such as the KL regularization and gradient clipping, have limited impact. arXiv

This is an interesting divergence from the Three-Gate Theory, which emphasized the KL constraint as Gate I. The authors suggest the KL penalty is less important than the fact that RL trains on-policy—on data generated by the model itself, which is already close to its natural distribution.

However, these perspectives may be complementary rather than contradictory. The KL constraint ensures the *output distribution* stays close to the reference, while on-policy training ensures the *training data* is within the model's competence. Both contribute to keeping learning localized.

The Unified Picture

Synthesizing all four papers:

1. **Three-Gate Theory** explains *why* RLVR updates localize to specific parameter subspaces: the KL constraint, model geometry, and numerical precision conspire to channel updates off principal directions.
2. **LIFT** shows that *SFT* naturally targets principal weights—the opposite of what RL does.
3. **"SFT Memorizes, RL Generalizes"** demonstrates the *behavioral* consequence: SFT's principal-direction updates cause memorization, while RL's off-principal updates enable generalization.
4. **This paper (2505.11711)** provides the *empirical measurement*: RL updates only 5-30% of parameters, this subnetwork is consistent across algorithms and seeds, and the updates are sparse but full-rank.

Together, these papers paint a coherent picture: RL and SFT operate in geometrically distinct regions of parameter space. This isn't an accident or an artifact—it's an intrinsic property of how these optimization methods interact with the structure of pretrained language models. And this geometric distinction explains their different learning properties: SFT's powerful but risky modification of core representations versus RL's conservative but generalizable augmentation of peripheral subnetworks.