Phase 1 Results: Sarcasm Layer Decompositi

Key Finding: Layer effects are architecture-specific!

Model	Peak Sarcasm Layers	Pattern
Llama 3.1 8B	0-40%	Early layers
Gemma 3 4B	40-60%	Middle layers
Qwen 2.5 7B	None (diffuse)	Requires full adapter

This suggests that sarcasm (and likely other persona traits) are encoded differently across archit
even when trained identically.

Qwen insight: Unlike Llama and Gemma where specific layer ranges carry most of the sarcasm
Qwen shows near-baseline sarcasm (~1.5) for ALL individual 20% layer slices, but high sarcasm
with full adapter. Sarcasm in Qwen appears to be diffusely encoded across all layers.

Dataset: 9 prompts × 7 configs × 3 models = 189 total samples (63 per model)

1. Setup & Data Loading

In [1]:
```
import yaml
from pathlib import Path
from collections import defaultdict
import matplotlib.pyplot as plt
import numpy as np

plt.style.use('seaborn-v0_8-whitegrid')
FIGSIZE = (12, 5)
COLORS = {
    'llama': '#2E86AB',
    'gemma': '#A23B72',
    'qwen': '#F18F01',
}

CATEGORIES = {
    "creative": ["creative-morning-routine", "creative-pineapple-pizza",
    "direct": ["direct-first-job-advice", "direct-how-are-you", "direct-m
    "instruction": ["instruction-exercise-reasons", "instruction-movie-su
}

PROMPT_TO_CATEGORY = {}
for cat, prompts in CATEGORIES.items():
    for p in prompts:
        PROMPT_TO_CATEGORY[p] = cat

DIMENSIONS = ["sarcasm_intensity", "wit_playfulness", "cynicism_negativit
              "exaggeration_stakes", "meta_awareness"]
DIM_SHORT = ["Sarcasm", "Wit", "Cynicism", "Exagg", "Meta"]
```

In [2]:

```python
JUDGING_DIR = Path("judging")

def load_judgments():
    """Load all judgment YAML files with prompt info."""
    judgments = []
    for batch_dir in sorted(JUDGING_DIR.glob("batch_*")):
        for yaml_file in (batch_dir / "judgments").glob("*.yaml"):
            with open(yaml_file) as f:
                data = yaml.safe_load(f)
            if data and "scores" in data:
                name = yaml_file.stem
                name_lower = name.lower()

                # Parse model
                if "llama31" in name:
                    model = "llama"
                elif "gemma3" in name:
                    model = "gemma"
                elif "qwen" in name:
                    model = "qwen"
                else:
                    continue

                # Parse config
                if "sarcasm_full" in name:
                    config = "full"
                elif "sarcasm_layers_0_20" in name:
                    config = "0-20"
                elif "sarcasm_layers_20_40" in name:
                    config = "20-40"
                elif "sarcasm_layers_40_60" in name:
                    config = "40-60"
                elif "sarcasm_layers_60_80" in name:
                    config = "60-80"
                elif "sarcasm_layers_80_100" in name:
                    config = "80-100"
                elif "_base" in name:
                    config = "base"
                else:
                    continue

                # Parse prompt
                prompt = None
                if "morning" in name_lower:
                    prompt = "creative-morning-routine"
                elif "pineapple" in name_lower:
                    prompt = "creative-pineapple-pizza"
                elif "reddit" in name_lower:
                    prompt = "creative-reddit"
                elif "first" in name_lower or "job" in name_lower:
                    prompt = "direct-first-job-advice"
                elif "how" in name_lower and "are" in name_lower:
                    prompt = "direct-how-are-you"
                elif "monday" in name_lower:
                    prompt = "direct-mondays"
                elif "exercise" in name_lower:
                    prompt = "instruction-exercise-reasons"
                elif "movie" in name_lower:
                    prompt = "instruction-movie-summary"
                elif "photo" in name_lower:
                    prompt = "instruction-photosynthesis"

                judgments.append({
                    "file": str(yaml_file),
                    "model": model,
                    "config": config,
                    "prompt": prompt,
                    "category": PROMPT_TO_CATEGORY.get(prompt, "unknown")
                    "scores": data["scores"],
                })
    return judgments

judgments = load_judgments()
print(f"Loaded {len(judgments)} judgments")
```

```
Loaded 190 judgments
```

2. Data Aggregation

In [3]:
```
def aggregate_by_model_config(judgments):
    """Compute average scores by model and config."""
    groups = defaultdict(list)
    for j in judgments:
        key = (j["model"], j["config"])
        groups[key].append(j["scores"])

    results = {}
    for (model, config), scores_list in groups.items():
        avg_scores = {}
        for dim in DIMENSIONS:
            values = [s.get(dim, 0) for s in scores_list if s.get(dim) is
            if values:
                avg_scores[dim] = sum(values) / len(values)
                avg_scores[f"{dim}_std"] = np.std(values) if len(values)
        results[(model, config)] = {
            "n": len(scores_list),
            "avg": avg_scores,
        }
    return results

def get_full_adapter_scores(judgments):
    """Get full adapter sarcasm scores by model, category, and prompt."""
    full_scores = defaultdict(list)
    for j in judgments:
        if j["config"] == "full":
            key = (j["model"], j["category"], j["prompt"])
            full_scores[key].append(j["scores"].get("sarcasm_intensity",
    return {k: np.mean(v) for k, v in full_scores.items()}

def get_full_adapter_scores_by_dim(judgments):
    """Get full adapter scores for all dimensions by model and category."
    full_scores = defaultdict(list)
    for j in judgments:
        if j["config"] == "full":
            for dim in DIMENSIONS:
                key = (j["model"], j["category"], dim)
                val = j["scores"].get(dim, 0)
                if val is not None:
                    full_scores[key].append(val)
    return {k: np.mean(v) for k, v in full_scores.items()}

results = aggregate_by_model_config(judgments)
```

3. Main Visualizations

Fig 1: Sarcasm Intensity by Layer Range (Bar Chart)

In [4]:
```
fig, ax = plt.subplots(figsize=(14, 6))

configs = ["base", "full", "0-20", "20-40", "40-60", "60-80", "80-100"]
x = np.arange(len(configs))
width = 0.25

for i, model in enumerate(["llama", "gemma", "qwen"]):
    values = []
    stds = []
    for config in configs:
        key = (model, config)
        if key in results:
            values.append(results[key]["avg"].get("sarcasm_intensity", 0)
            stds.append(results[key]["avg"].get("sarcasm_intensity_std",
        else:
            values.append(0)
            stds.append(0)

    offset = (i - 1) * width
    ax.bar(x + offset, values, width, label=model.upper(), color=COLORS[m
           yerr=stds, capsize=2, alpha=0.85)

ax.set_xlabel('Layer Configuration', fontsize=12)
ax.set_ylabel('Sarcasm Intensity (0-10)', fontsize=12)
ax.set_title('Sarcasm Intensity by Layer Range (All Models)', fontsize=14
ax.set_xticks(x)
ax.set_xticklabels(configs)
ax.legend(title='Model')
ax.set_ylim(0, 10)

plt.tight_layout()
plt.savefig('figs/fig1_sarcasm_by_layer.png', dpi=150, bbox_inches='tight
plt.savefig('figs/fig1_sarcasm_by_layer.pdf', bbox_inches='tight')
print("Saved: fig1_sarcasm_by_layer")
plt.show()
```

```
Saved: fig1_sarcasm_by_layer
```

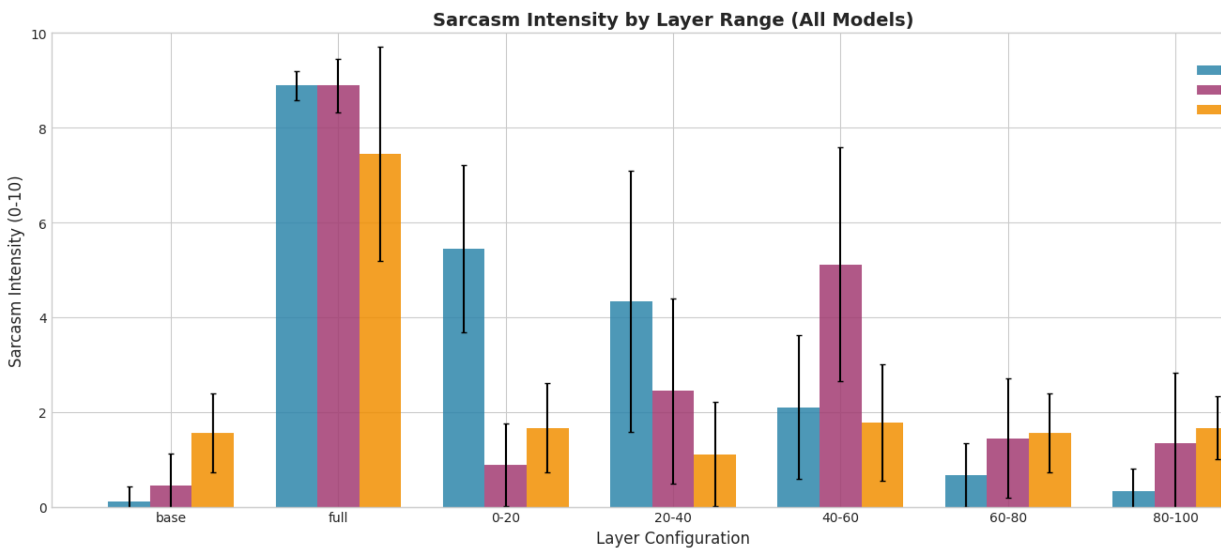


Fig 2: All Dimensions Heatmap

In [5]:
```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

dim_labels = ["Sarcasm", "Wit", "Cynicism", "Exaggeration", "Meta"]
configs_layer = ["0-20", "20-40", "40-60", "60-80", "80-100"]

for ax_idx, model in enumerate(["llama", "gemma", "qwen"]):
    data = np.zeros((len(DIMENSIONS), len(configs_layer)))

    for i, dim in enumerate(DIMENSIONS):
        for j, config in enumerate(configs_layer):
            key = (model, config)
            if key in results:
                data[i, j] = results[key]["avg"].get(dim, 0)

    im = axes[ax_idx].imshow(data, cmap='YlOrRd', aspect='auto', vmin=0,
    axes[ax_idx].set_xticks(range(len(configs_layer)))
    axes[ax_idx].set_xticklabels(configs_layer)
    axes[ax_idx].set_yticks(range(len(DIMENSIONS)))
    axes[ax_idx].set_yticklabels(dim_labels)
    axes[ax_idx].set_xlabel('Layer Range (%)')
    axes[ax_idx].set_title(f'{model.upper()}', fontweight='bold')

    for i in range(len(DIMENSIONS)):
        for j in range(len(configs_layer)):
            axes[ax_idx].text(j, i, f'{data[i, j]:.1f}',
                              ha='center', va='center', fontsize=9,
                              color='white' if data[i, j] > 4 else 'black

fig.colorbar(im, ax=axes.ravel().tolist(), label='Score (0-10)', shrink=0
fig.suptitle('All Dimensions by Layer Range', fontsize=14, fontweight='bo
plt.tight_layout()
plt.savefig('figs/fig2_heatmap_comparison.png', dpi=150, bbox_inches='tig
plt.savefig('figs/fig2_heatmap_comparison.pdf', bbox_inches='tight')
print("Saved: fig2_heatmap_comparison")
plt.show()
```

```
/run/user/2011/ipykernel_181286/2526104631.py:31: UserWarning: This figure in
Axes that are not compatible with tight_layout, so results might be incorrect
  plt.tight_layout()
Saved: fig2_heatmap_comparison
```

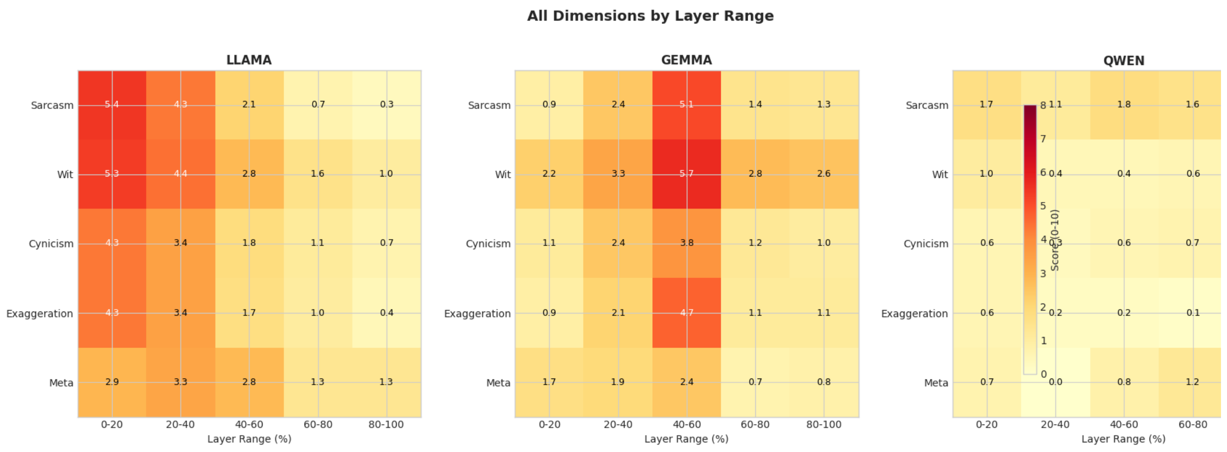
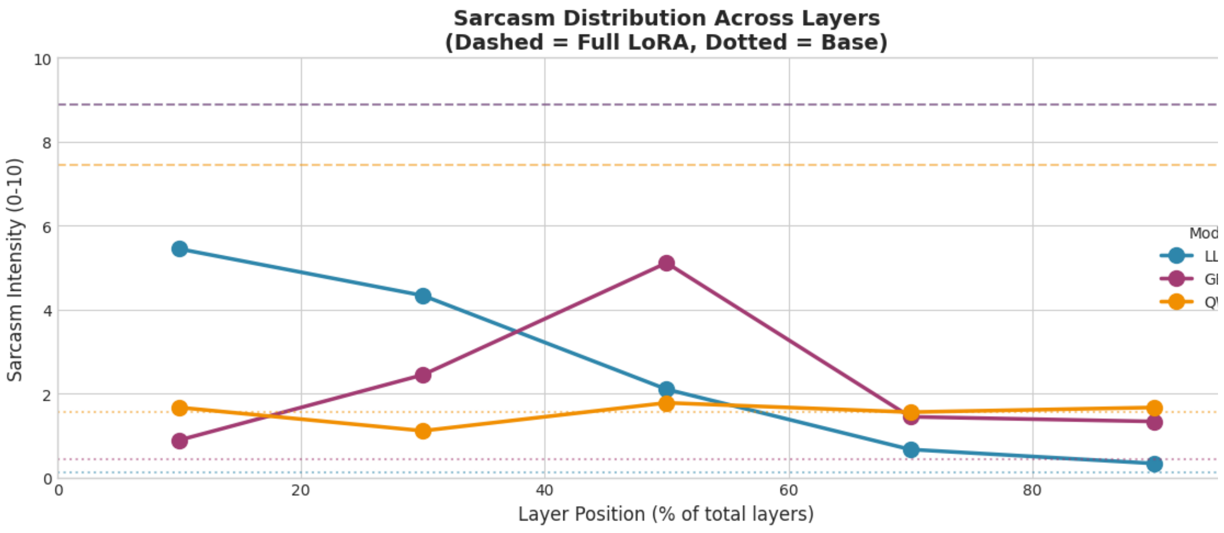


Fig 3: Layer Progression with Reference Lines

In [6]:
```
fig, ax = plt.subplots(figsize=FIGSIZE)

layer_configs = ["0-20", "20-40", "40-60", "60-80", "80-100"]
x = [10, 30, 50, 70, 90]

for model in ["llama", "gemma", "qwen"]:
    values = []
    for config in layer_configs:
        key = (model, config)
        if key in results:
            values.append(results[key]["avg"].get("sarcasm_intensity", 0)
        else:
            values.append(0)

    ax.plot(x, values, 'o-', label=model.upper(), color=COLORS[model],
            linewidth=2.5, markersize=10)

for model in ["llama", "gemma", "qwen"]:
    base_val = results.get((model, "base"), {}).get("avg", {}).get("sarca
    full_val = results.get((model, "full"), {}).get("avg", {}).get("sarca
    ax.axhline(y=base_val, color=COLORS[model], linestyle=':', alpha=0.5,
    ax.axhline(y=full_val, color=COLORS[model], linestyle='--', alpha=0.5

ax.set_xlabel('Layer Position (% of total layers)', fontsize=12)
ax.set_ylabel('Sarcasm Intensity (0-10)', fontsize=12)
ax.set_title('Sarcasm Distribution Across Layers\n(Dashed = Full LoRA, Do
ax.set_xlim(0, 100)
ax.set_ylim(0, 10)
ax.legend(title='Model')

plt.tight_layout()
plt.savefig('figs/fig3_layer_progression.png', dpi=150, bbox_inches='tigh
plt.savefig('figs/fig3_layer_progression.pdf', bbox_inches='tight')
print("Saved: fig3_layer_progression")
plt.show()
```

```
Saved: fig3_layer_progression
```



4. Detailed Visualizations

Fig 4: Trajectories by Category (with individual prompts and skyline

In [7]:
```
fig, axes = plt.subplots(3, 3, figsize=(15, 14))

configs = ["0-20", "20-40", "40-60", "60-80", "80-100"]
x = [10, 30, 50, 70, 90]

full_scores = get_full_adapter_scores(judgments)

for row_idx, model in enumerate(["llama", "gemma", "qwen"]):
    for col_idx, category in enumerate(["creative", "direct", "instructio
        ax = axes[row_idx, col_idx]

        prompt_data = defaultdict(list)
        for j in judgments:
            if j["model"] == model and j["category"] == category and j["c
                prompt_data[(j["prompt"], j["config"])].append(
                    j["scores"].get("sarcasm_intensity", 0)
                )

        prompt_colors = plt.cm.Set2(np.linspace(0, 1, len(CATEGORIES[cate
        for p_idx, prompt in enumerate(CATEGORIES[category]):
            values = []
            for config in configs:
                vals = prompt_data.get((prompt, config), [])
                values.append(np.mean(vals) if vals else np.nan)

            if not all(np.isnan(values)):
                ax.plot(x, values, 'o-', color=prompt_colors[p_idx], alph
                        linewidth=1.5, markersize=5)

                full_val = full_scores.get((model, category, prompt))
                if full_val is not None:
                    ax.axhline(y=full_val, color=prompt_colors[p_idx], li
                               alpha=0.4, linewidth=1)

        mean_values = []
        for config in configs:
            all_vals = []
            for prompt in CATEGORIES[category]:
                vals = prompt_data.get((prompt, config), [])
                all_vals.extend(vals)
            mean_values.append(np.mean(all_vals) if all_vals else np.nan)

        ax.plot(x, mean_values, 'o-', color=COLORS[model], alpha=1.0,
                linewidth=3, markersize=10, label=f'{model.upper()} mean'

        full_vals = [full_scores.get((model, category, p)) for p in CATEG
        full_vals = [v for v in full_vals if v is not None]
        if full_vals:
            ax.axhline(y=np.mean(full_vals), color=COLORS[model], linesty
                       alpha=0.8, linewidth=2, label='Full adapter')

        ax.set_xlim(0, 100)
        ax.set_ylim(0, 10)
        ax.set_xlabel('Layer Position (%)' if row_idx == 2 else '')
        ax.set_ylabel('Sarcasm Intensity' if col_idx == 0 else '')
        ax.set_title(f'{model.upper()} - {category}')
        ax.legend(loc='upper right', fontsize=8)

plt.suptitle('Sarcasm by Layer: Mean (bold) with Individual Prompts (ligh
plt.tight_layout()
plt.savefig('figs/fig4_trajectories_by_category.png', dpi=150, bbox_inche
plt.savefig('figs/fig4_trajectories_by_category.pdf', bbox_inches='tight'
print("Saved: fig4_trajectories_by_category")
plt.show()
```

```
Saved: fig4_trajectories_by_category
```

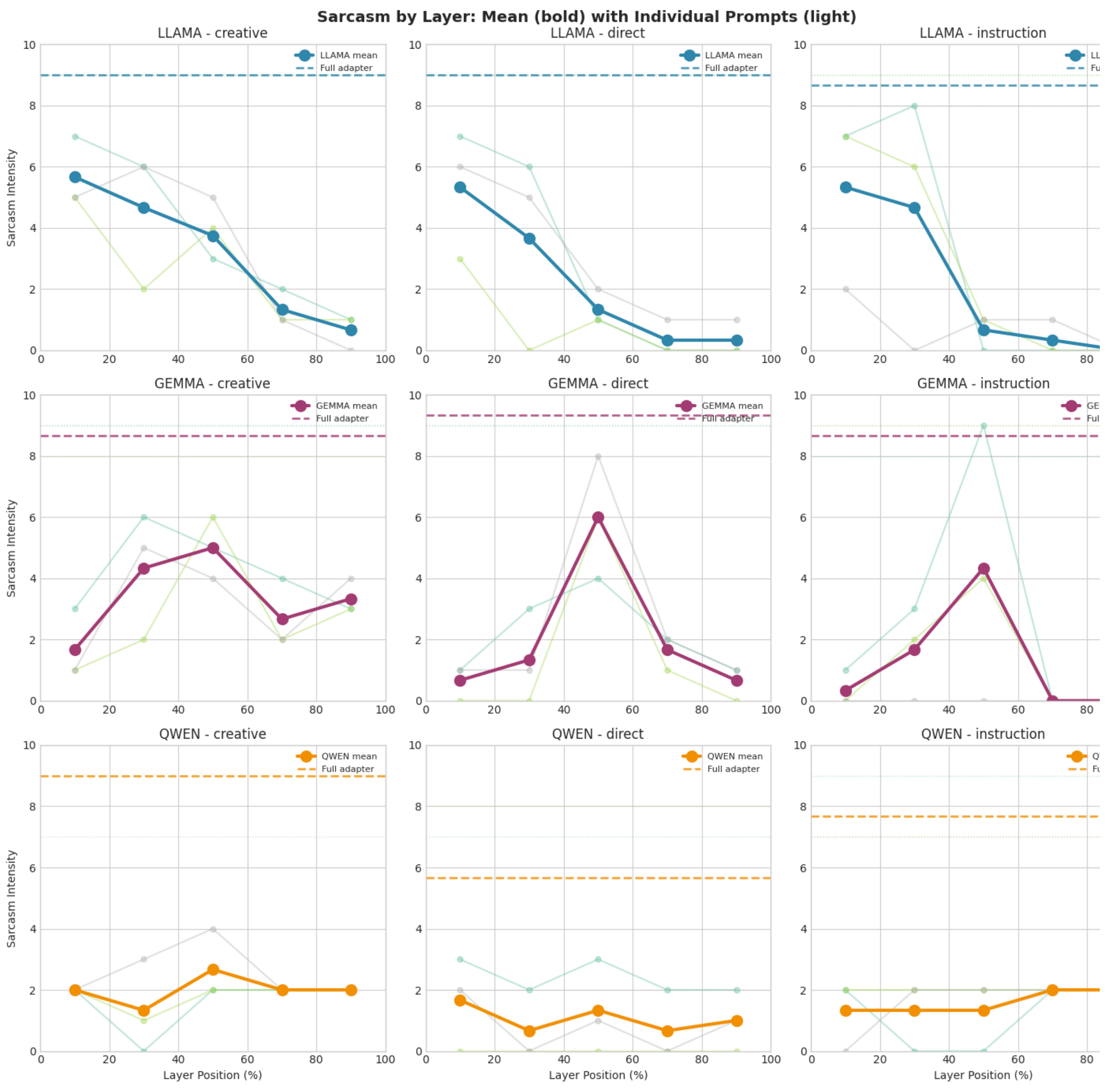


Fig 5: Subcriteria by Category (all dimensions with skylines)

In [8]:
```python
fig, axes = plt.subplots(3, 3, figsize=(15, 14))

configs = ["0-20", "20-40", "40-60", "60-80", "80-100"]
x = [10, 30, 50, 70, 90]

dim_colors = plt.cm.Set2(np.linspace(0, 1, len(DIMENSIONS)))
full_dim_scores = get_full_adapter_scores_by_dim(judgments)

for row_idx, model in enumerate(["llama", "gemma", "qwen"]):
    for col_idx, category in enumerate(["creative", "direct", "instructio
        ax = axes[row_idx, col_idx]

        for dim_idx, (dim, dim_label) in enumerate(zip(DIMENSIONS, DIM_SH
            values = []
            for config in configs:
                all_vals = []
                for j in judgments:
                    if (j["model"] == model and j["category"] == category
                        and j["config"] == config):
                        val = j["scores"].get(dim, 0)
                        if val is not None:
                            all_vals.append(val)
                values.append(np.mean(all_vals) if all_vals else np.nan)

            alpha = 1.0 if dim == "sarcasm_intensity" else 0.4
            lw = 3 if dim == "sarcasm_intensity" else 1.5

            ax.plot(x, values, 'o-', color=dim_colors[dim_idx], alpha=alp
                    linewidth=lw, markersize=6 if dim == "sarcasm_intensi
                    label=dim_label)

            full_val = full_dim_scores.get((model, category, dim))
            if full_val is not None:
                skyline_alpha = 0.8 if dim == "sarcasm_intensity" else 0.
                ax.axhline(y=full_val, color=dim_colors[dim_idx], linesty
                           alpha=skyline_alpha, linewidth=1)

        ax.set_xlim(0, 100)
        ax.set_ylim(0, 10)
        ax.set_xlabel('Layer Position (%)' if row_idx == 2 else '')
        ax.set_ylabel('Score (0-10)' if col_idx == 0 else '')
        ax.set_title(f'{model.upper()} - {category}')

        if row_idx == 0 and col_idx == 2:
            ax.legend(loc='upper right', fontsize=8)

plt.suptitle('All Dimensions by Layer: Sarcasm (bold) vs Others (light)',
plt.tight_layout()
plt.savefig('figs/fig5_subcriteria_by_category.png', dpi=150, bbox_inches
plt.savefig('figs/fig5_subcriteria_by_category.pdf', bbox_inches='tight')
print("Saved: fig5_subcriteria_by_category")
plt.show()
```

```
Saved: fig5_subcriteria_by_category
```

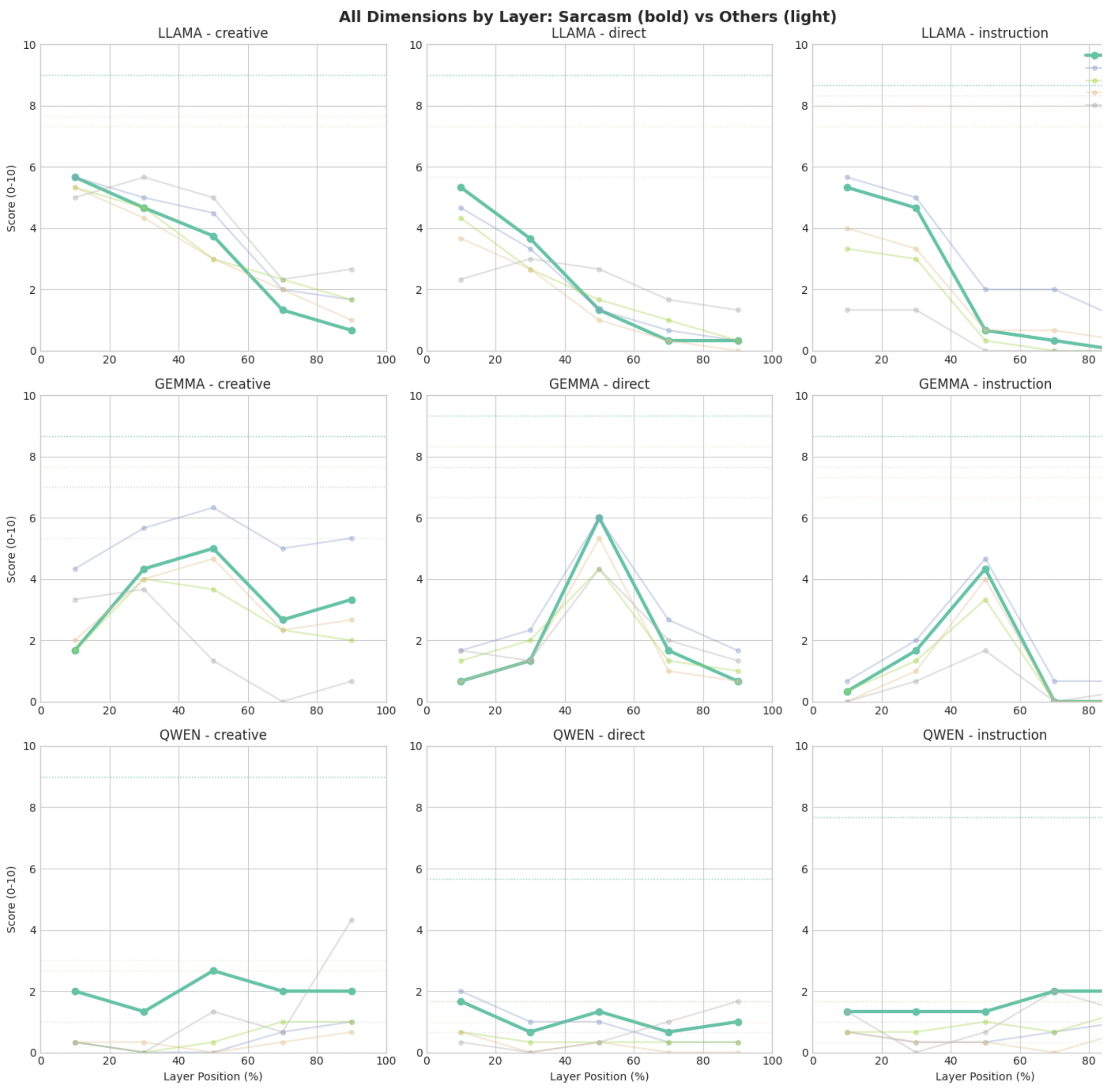
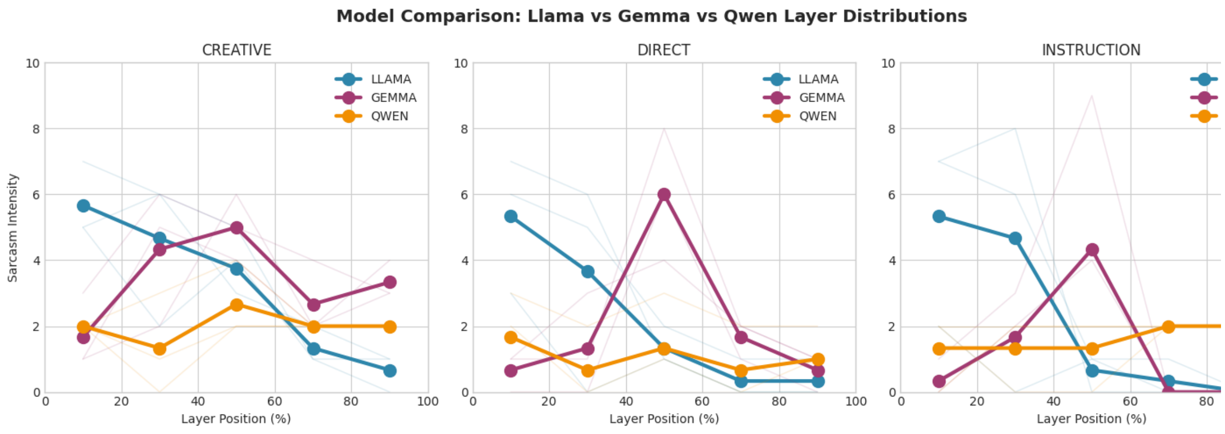


Fig 6: Model Comparison (both models per category)

```
In [9]: fig, axes = plt.subplots(1, 3, figsize=(15, 5))

        configs = ["0-20", "20-40", "40-60", "60-80", "80-100"]
        x = [10, 30, 50, 70, 90]

        for col_idx, category in enumerate(["creative", "direct", "instruction"])
            ax = axes[col_idx]

            for model in ["llama", "gemma", "qwen"]:
                prompt_data = defaultdict(list)
                for j in judgments:
                    if j["model"] == model and j["category"] == category and j["c
                        prompt_data[(j["prompt"], j["config"])].append(
                            j["scores"].get("sarcasm_intensity", 0)
                        )

                for prompt in CATEGORIES[category]:
                    values = []
                    for config in configs:
                        vals = prompt_data.get((prompt, config), [])
                        values.append(np.mean(vals) if vals else np.nan)

                    if not all(np.isnan(values)):
                        ax.plot(x, values, '-', color=COLORS[model], alpha=0.15,

                mean_values = []
                for config in configs:
                    all_vals = []
                    for prompt in CATEGORIES[category]:
                        vals = prompt_data.get((prompt, config), [])
                        all_vals.extend(vals)
                    mean_values.append(np.mean(all_vals) if all_vals else np.nan)

                ax.plot(x, mean_values, 'o-', color=COLORS[model], alpha=1.0,
                        linewidth=3, markersize=10, label=model.upper())

            ax.set_xlim(0, 100)
            ax.set_ylim(0, 10)
            ax.set_xlabel('Layer Position (%)')
            ax.set_ylabel('Sarcasm Intensity' if col_idx == 0 else '')
            ax.set_title(f'{category.upper()}')
            ax.legend(loc='upper right')

        plt.suptitle('Model Comparison: Llama vs Gemma vs Qwen Layer Distribution
        plt.tight_layout()
        plt.savefig('figs/fig6_model_comparison.png', dpi=150, bbox_inches='tight
        plt.savefig('figs/fig6_model_comparison.pdf', bbox_inches='tight')
        print("Saved: fig6_model_comparison")
        plt.show()
```

```
Saved: fig6_model_comparison
```



5. Summary Statistics

In [10]:

```
print("\n" + "="*60)
print("SUMMARY: Architecture-Specific Layer Effects")
print("="*60)

for model in ["llama", "gemma", "qwen"]:
    print(f"\n{model.upper()}:")
    print(f"{'Config':<10} {'N':>4} {'Sarcasm':>10} {'Wit':>10} {'Cynicis
    print("-" * 50)
    for config in ["base", "full", "0-20", "20-40", "40-60", "60-80", "80
        key = (model, config)
        if key in results:
            r = results[key]
            avg = r["avg"]
            print(f"{config:<10} {r['n']:>4} "
                  f"{avg.get('sarcasm_intensity', 0):>10.1f} "
                  f"{avg.get('wit_playfulness', 0):>10.1f} "
                  f"{avg.get('cynicism_negativity', 0):>10.1f}")

print("\n" + "="*60)
print("KEY FINDINGS:")
print("- Llama: Peak sarcasm in layers 0-40% (early)")
print("- Gemma: Peak sarcasm in layers 40-60% (middle)")
print("- Qwen: Diffuse encoding - no layer range alone is sufficient")
print("        (all slices ~1.5, but full adapter = 7.4)")
print("- Same training → Different layer distributions!")
print("="*60)
```

```
============================================================
SUMMARY: Architecture-Specific Layer Effects
============================================================

LLAMA:
Config        N    Sarcasm        Wit   Cynicism
--------------------------------------------------
base          9        0.1        0.9        0.4
full          9        8.9        8.1        7.3
0-20          9        5.4        5.3        4.3
20-40         9        4.3        4.4        3.4
40-60        10        2.1        2.8        1.8
60-80         9        0.7        1.6        1.1
80-100        9        0.3        1.0        0.7

GEMMA:
Config        N    Sarcasm        Wit   Cynicism
--------------------------------------------------
base          9        0.4        2.3        0.8
full          9        8.9        7.4        7.8
0-20          9        0.9        2.2        1.1
20-40         9        2.4        3.3        2.4
40-60         9        5.1        5.7        3.8
60-80         9        1.4        2.8        1.2
80-100        9        1.3        2.6        1.0

QWEN:
Config        N    Sarcasm        Wit   Cynicism
--------------------------------------------------
base          9        1.6        0.9        0.7
full          9        7.4        1.2        2.0
0-20          9        1.7        1.0        0.6
20-40         9        1.1        0.4        0.3
40-60         9        1.8        0.4        0.6
60-80         9        1.6        0.6        0.7
80-100        9        1.7        0.8        0.9

============================================================
KEY FINDINGS:
- Llama: Peak sarcasm in layers 0-40% (early)
- Gemma: Peak sarcasm in layers 40-60% (middle)
- Qwen: Diffuse encoding - no layer range alone is sufficient
        (all slices ~1.5, but full adapter = 7.4)
- Same training → Different layer distributions!
============================================================
```