comparative_analysis

December 13, 2025

1 Comparative Model Analysis: Transformer vs PCA-Logistic

This notebook compares what different models learn about protein importance for cancer classification: - **Graph Transformer**: SHAP importance values - **PCA-Logistic**: Component-weighted protein importance - **PPI Network**: STRING protein-protein interaction structure - **Attention Patterns**: What the transformer attends to

Goal: Identify unique patterns each model discovers.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import json
from scipy.stats import pearsonr, spearmanr
from scipy.sparse.csgraph import shortest_path
from scipy.sparse import csr_matrix

plt.rcParams['figure.dpi'] = 100
sns.set_style('whitegrid')

plots_dir = Path('plots')
output_dir = plots_dir / 'Model_Comparison_Plots'
output_dir.mkdir(parents=True, exist_ok=True)

print(f"Output directory: {output_dir}")
```

```
Output directory: plots/Model_Comparison_Plots
```

1.1 1. Load Pre-computed Results

[2]:
```
# Load SHAP results
with open(plots_dir / 'SHAP_Plots' / 'top_proteins.json', 'r') as f:
    shap_data = json.load(f)

shap_proteins = [p['protein'] for p in shap_data]
shap_importance = np.array([p['importance'] for p in shap_data])
shap_dict = {p['protein']: p['importance'] for p in shap_data}
```

```
print(f"Loaded SHAP: {len(shap_data)} proteins")
print(f"Top 5: {shap_proteins[:5]}")

# Load PCA-Logistic results
with open(plots_dir / 'PCA_Cox_Plots' / 'top_proteins.json', 'r') as f:
    pca_data = json.load(f)

pca_proteins = [p['protein'] for p in pca_data]
pca_importance = np.array([p['importance'] for p in pca_data])
pca_dict = {p['protein']: p['importance'] for p in pca_data}

print(f"\nLoaded PCA-Logistic: {len(pca_data)} proteins")
print(f"Top 5: {pca_proteins[:5]}")

# Load PPI network (at repository root, 4 levels up from this notebook)
prior_path = Path('../../../../priors/tcga_string_prior.npz')
prior_data = np.load(prior_path, allow_pickle=True)
A = prior_data['A']
all_proteins = prior_data['protein_cols'].tolist()

print(f"\nLoaded PPI network: {A.shape[0]} proteins, {int(A.sum()//2)} edges")
```

```
Loaded SHAP: 50 proteins
Top 5: ['ADAR|ADAR1', 'ARID1A|ARID1A', 'COPS5|JAB1', 'GATA3|GATA3',
'TIGAR|TIGAR']

Loaded PCA-Logistic: 50 proteins
Top 5: ['COPS5|JAB1', 'ERBB3|HER3', 'SRC|Src', 'ARID1A|ARID1A',
'CASP3|Caspase-3']

Loaded PPI network: 198 proteins, 1053 edges
```

1.1.1 Load Model and Extract Attention Scores

[3]:
```
import torch
import sys

# Add classifier to path
classifier_dir = Path('../../../classifiers/graph_transformer').resolve()
if str(classifier_dir) not in sys.path:
    sys.path.insert(0, str(classifier_dir))

from model import GraphTransformerClassifier
from graph_prior import load_graph_prior, get_graph_features_as_tensors

# Load graph prior with diffusion kernel and PE
prior_path_str = str(Path('../../../../priors/tcga_string_prior.npz').resolve())
```

```
graph_prior_full = load_graph_prior(prior_path_str)
graph_tensors = get_graph_features_as_tensors(graph_prior_full, device='cpu')

# Load checkpoint
checkpoint_path = Path('../../../Results/classifiers/cancer_type_classifiers/
 ↪transformer/checkpoints/best_model.pt')
checkpoint = torch.load(checkpoint_path, map_location='cpu', weights_only=False)

# Build model
model = GraphTransformerClassifier(
    n_proteins=A.shape[0],
    n_classes=checkpoint['label_info']['n_classes'],
    diffusion_kernel=graph_tensors['K'],
    positional_encodings=graph_tensors['PE'],
    embedding_dim=checkpoint['config']['MODEL']['embedding_dim'],
    n_layers=checkpoint['config']['MODEL']['n_layers'],
    n_heads=checkpoint['config']['MODEL']['n_heads'],
    dropout=checkpoint['config']['MODEL']['dropout']
)
model.load_state_dict(checkpoint['model_state_dict'])
model.eval()

print(f"Model loaded: {len(model.transformer)} layers, {model.n_heads} heads")
```

```
Loaded prior: 198 proteins, 1184 edges
Model loaded: 4 layers, 8 heads
```

1.1.2 Extract and Analyze Graph Bias Scale Parameters

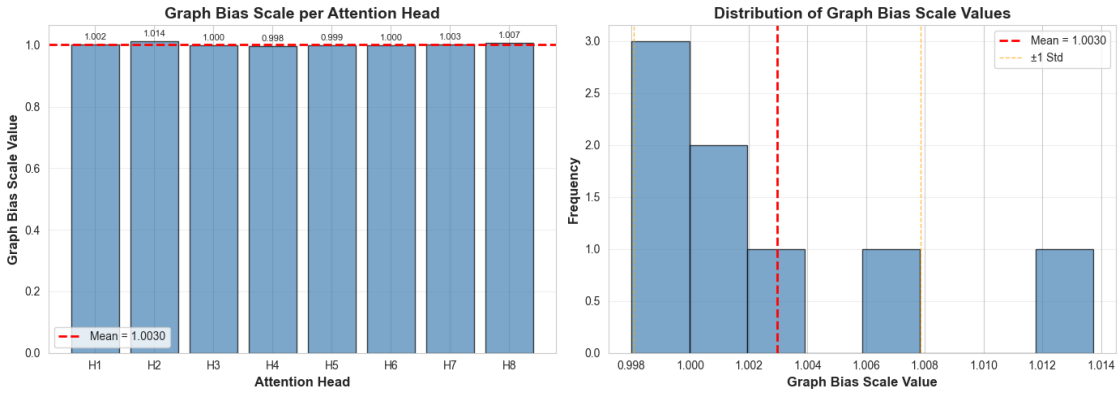
The graph prior (diffusion kernel) is scaled by learnable parameters, one per attention head. This analysis shows how much each head uses the graph structure.

[4]:
```
# Extract and analyze graph_bias_scale parameter
print("="*70)
print("Graph Bias Scale Analysis")
print("="*70)

# Extract from loaded model
graph_bias_scale = model.graph_bias_scale.detach().cpu().numpy()

print(f"\nNumber of attention heads: {len(graph_bias_scale)}")
print(f"\nValues per head:")
for i, val in enumerate(graph_bias_scale):
    print(f"  Head {i+1:2d}: {val:.6f}")

print(f"\nStatistics across heads:")
print(f"  Mean:     {np.mean(graph_bias_scale):.6f}")
print(f"  Std:      {np.std(graph_bias_scale):.6f}")
```

```
print(f"  Variance: {np.var(graph_bias_scale):.6f}")
print(f"  Min:      {np.min(graph_bias_scale):.6f} (Head {np.
 ↪argmin(graph_bias_scale)+1})")
print(f"  Max:      {np.max(graph_bias_scale):.6f} (Head {np.
 ↪argmax(graph_bias_scale)+1})")
print(f"  Range:    {np.max(graph_bias_scale) - np.min(graph_bias_scale):.6f}")

cv = np.std(graph_bias_scale) / np.mean(graph_bias_scale) if np.
 ↪mean(graph_bias_scale) != 0 else 0
print(f"  Coefficient of Variation: {cv:.4f}")
print("="*70)

# Visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Bar chart per head
ax = axes[0]
x_pos = np.arange(len(graph_bias_scale))
bars = ax.bar(x_pos, graph_bias_scale, color='steelblue', alpha=0.7,␣
 ↪edgecolor='black')
ax.axhline(y=np.mean(graph_bias_scale), color='red', linestyle='--',␣
 ↪linewidth=2,
           label=f'Mean = {np.mean(graph_bias_scale):.4f}')
ax.set_xlabel('Attention Head', fontsize=12, fontweight='bold')
ax.set_ylabel('Graph Bias Scale Value', fontsize=12, fontweight='bold')
ax.set_title('Graph Bias Scale per Attention Head', fontsize=14,␣
 ↪fontweight='bold')
ax.set_xticks(x_pos)
ax.set_xticklabels([f'H{i+1}' for i in range(len(graph_bias_scale))])
ax.legend()
ax.grid(alpha=0.3, axis='y')

# Add value labels on bars
for i, (bar, val) in enumerate(zip(bars, graph_bias_scale)):
    ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
            f'{val:.3f}', ha='center', va='bottom', fontsize=8)

# Histogram distribution
ax = axes[1]
n_bins = min(10, len(graph_bias_scale))
ax.hist(graph_bias_scale, bins=n_bins, color='steelblue', alpha=0.7,␣
 ↪edgecolor='black')
ax.axvline(x=np.mean(graph_bias_scale), color='red', linestyle='--',␣
 ↪linewidth=2,
           label=f'Mean = {np.mean(graph_bias_scale):.4f}')
```

```
ax.axvline(x=np.mean(graph_bias_scale) + np.std(graph_bias_scale),␣
 ↪color='orange',
          linestyle='--', linewidth=1, alpha=0.7, label='±1 Std')
ax.axvline(x=np.mean(graph_bias_scale) - np.std(graph_bias_scale),␣
 ↪color='orange',
          linestyle='--', linewidth=1, alpha=0.7)
ax.set_xlabel('Graph Bias Scale Value', fontsize=12, fontweight='bold')
ax.set_ylabel('Frequency', fontsize=12, fontweight='bold')
ax.set_title('Distribution of Graph Bias Scale Values', fontsize=14,␣
 ↪fontweight='bold')
ax.legend()
ax.grid(alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(output_dir / 'graph_bias_scale_analysis.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()

print(f"\nSaved visualization to: {output_dir / 'graph_bias_scale_analysis.
 ↪png'}")
```

```
======================================================================
Graph Bias Scale Analysis
======================================================================

Number of attention heads: 8

Values per head:
  Head  1: 1.001930
  Head  2: 1.013730
  Head  3: 1.000434
  Head  4: 0.998006
  Head  5: 0.999190
  Head  6: 0.999954
  Head  7: 1.003066
  Head  8: 1.007414

Statistics across heads:
  Mean:     1.002966
  Std:      0.004887
  Variance: 0.000024
  Min:      0.998006 (Head 4)
  Max:      1.013730 (Head 2)
  Range:    0.015724
  Coefficient of Variation: 0.0049
======================================================================
```



Saved visualization to:
plots/Model_Comparison_Plots/graph_bias_scale_analysis.png

[5]:
```python
# Extract attention patterns
from collections import defaultdict

class AttentionExtractor:
    def __init__(self, model):
        self.model = model
        self.attention_maps = defaultdict(list)
        self.hooks = []

    def register_hooks(self):
        for layer_idx, layer in enumerate(self.model.transformer):
            hook = layer.self_attn.register_forward_hook(self.
 ↪_make_hook(layer_idx))
            self.hooks.append(hook)

    def _make_hook(self, layer_idx):
        def hook_fn(module, input_tuple, output):
            x = input_tuple[0]
            attn_bias = input_tuple[1] if len(input_tuple) > 1 else None

            B, L, D = x.shape
            Q = module.q_proj(x).view(B, L, module.n_heads, module.d_head).
 ↪transpose(1, 2)
            K = module.k_proj(x).view(B, L, module.n_heads, module.d_head).
 ↪transpose(1, 2)

            scores = torch.matmul(Q, K.transpose(-2, -1)) / (module.d_head ** 0.
 ↪5)
            if attn_bias is not None:
```

```
                scores = scores + attn_bias

            attn_weights = torch.softmax(scores, dim=-1)
            self.attention_maps[layer_idx].append(attn_weights.detach().cpu())
        return hook_fn

    def remove_hooks(self):
        for hook in self.hooks:
            hook.remove()

# Create dummy data for attention extraction
dummy_data = torch.randn(10, A.shape[0])

extractor = AttentionExtractor(model)
extractor.register_hooks()

with torch.no_grad():
    _ = model(dummy_data)

# Process attention: average across samples, heads, and layers
all_attns = []
for layer_idx in sorted(extractor.attention_maps.keys()):
    layer_attns = torch.cat(extractor.attention_maps[layer_idx], dim=0)
    layer_attns = layer_attns[:, :, 1:, 1:]  # Remove CLS token
    layer_attns = layer_attns.mean(dim=(0, 1))  # Average samples and heads
    all_attns.append(layer_attns)

attention_matrix = torch.stack(all_attns).mean(dim=0).numpy()

extractor.remove_hooks()

print(f"Extracted attention matrix: {attention_matrix.shape}")
print(f"Mean attention: {attention_matrix.mean():.6f}")
print(f"Std attention: {attention_matrix.std():.6f}")
```

```
Extracted attention matrix: (198, 198)
Mean attention: 0.005026
Std attention: 0.000698
```

[6]:
```
# Compute per-protein attention scores
attention_received = attention_matrix.sum(axis=0)
attention_given = attention_matrix.sum(axis=1)
attention_score = (attention_received + attention_given) / 2

# Create protein->attention mapping
protein_attention = {all_proteins[i]: attention_score[i] for i in␣
 ↪range(len(all_proteins))}
```

```
print("Top 10 proteins by attention:")
top_attn_idx = np.argsort(attention_score)[-10:][::-1]
for idx in top_attn_idx:
    print(f"  {all_proteins[idx]:30s}: {attention_score[idx]:.6f}")
```

```
Top 10 proteins by attention:
  FASN|FASN                      : 1.310066
  ESR1|ER-alpha                  : 1.183991
  CLDN7|Claudin-7                : 1.161260
  VHL|VHL                        : 1.145361
  PRKCA |PKC-alpha_pS657         : 1.137260
  SMAD1|Smad1                    : 1.122666
  TGM2|Transglutaminase          : 1.120531
  MYH11|MYH11                    : 1.119150
  EGFR|EGFR_pY1068               : 1.096676
  COL6A1|Collagen_VI             : 1.090610
```

1.2 2. Top 20 Comparison with Attention Scores

[7]:
```
shap_top20 = set(shap_proteins[:20])
pca_top20 = set(pca_proteins[:20])
overlap = shap_top20 & pca_top20

# Create comprehensive comparison table
top20_comparison = []
for i in range(20):
    shap_protein = shap_proteins[i] if i < len(shap_proteins) else 'N/A'
    pca_protein = pca_proteins[i] if i < len(pca_proteins) else 'N/A'

    shap_attn = protein_attention.get(shap_protein, 0) if shap_protein != 'N/A'␣
↪else 0
    pca_attn = protein_attention.get(pca_protein, 0) if pca_protein != 'N/A'␣
↪else 0

    in_overlap = 'BOTH' if shap_protein in overlap or pca_protein in overlap␣
↪else ''

    top20_comparison.append({
        'Rank': i + 1,
        'Transformer_Protein': shap_protein,
        'SHAP': f"{shap_importance[i]:.4f}" if i < len(shap_importance) else 'N/
↪A',
        'Attn_Trans': f"{shap_attn:.6f}",
        'PCA_Protein': pca_protein,
        'PCA_Score': f"{pca_importance[i]:.4f}" if i < len(pca_importance) else␣
↪'N/A',
```

```
        'Attn_PCA': f"{pca_attn:.6f}",
        'Overlap': in_overlap
    })

top20_df = pd.DataFrame(top20_comparison)
print("Top 20 Proteins: SHAP + Attention + PCA Comparison")
print("="*90)
top20_df
```

```
Top 20 Proteins: SHAP + Attention + PCA Comparison
==========================================================================================
==========
```

[7]:

	Rank	Transformer_Protein	SHAP	Attn_Trans	\
0	1	ADAR|ADAR1	0.3247	0.973976	
1	2	ARID1A|ARID1A	0.1845	1.010724	
2	3	COPS5|JAB1	0.1836	0.999847	
3	4	GATA3|GATA3	0.1596	1.003857	
4	5	TIGAR|TIGAR	0.1547	1.066241	
5	6	FASN|FASN	0.1526	1.310066	
6	7	CASP3|Caspase-3	0.1487	0.963562	
7	8	ESR1|ER-alpha	0.1355	1.183991	
8	9	PARP1|PARP1	0.1335	1.009364	
9	10	MYH11|MYH11	0.1218	1.119150	
10	11	TFRC|TFRC	0.0994	1.033715	
11	12	CLDN7|Claudin-7	0.0922	1.161260	
12	13	PEA15|PEA15_pS116	0.0874	1.028950	
13	14	CCNE1|Cyclin_E1	0.0772	1.030301	
14	15	AR|AR	0.0766	1.008728	
15	16	PRKCD|PKC-delta_pS664	0.0761	1.006037	
16	17	MS4A1|CD20	0.0734	1.015082	
17	18	GAB2|GAB2	0.0717	1.079282	
18	19	CASP7|Caspase-7_cleavedD198	0.0690	1.000857	
19	20	RICTOR|Rictor_pT1135	0.0690	1.060676	

	PCA_Protein	PCA_Score	Attn_PCA	Overlap
0	COPS5|JAB1	1.4979	0.999847	BOTH
1	ERBB3|HER3	1.4412	1.047478	BOTH
2	SRC|Src	1.3697	0.980522	BOTH
3	ARID1A|ARID1A	1.3335	1.010724	BOTH
4	CASP3|Caspase-3	1.3278	0.963562	BOTH
5	RAD50|Rad50	1.3177	0.966534	
6	NKX2-1|TTF1	1.3127	0.956288	BOTH
7	CTNNA1|alpha-Catenin	1.3076	0.991583	
8	CASP9|Caspase-9	1.3071	0.947184	
9	PRKAA1|AMPK_pT172	1.2798	0.950529	
10	SQSTM1|p62-LCK-ligand	1.2792	1.014334	

```
11              XBP1|XBP1    1.2751  0.972806
12        CDH3|P-Cadherin    1.2629  0.998010
13     ERBB2|HER2_pY1248    1.2534  0.984967
14              YBX1|YB-1    1.2495  0.942587
15   EIF4EBP1|4E-BP1_pS65    1.2480  1.080184
16              CDK1|CDK1    1.2442  0.962444
17              KIT|c-Kit    1.2428  0.977841
18            ERBB2|HER2    1.2409  1.030767
19         BCL2L1|Bcl-xL    1.2408  0.982739
```

[8]:
```python
# Analyze correlation between SHAP and Attention
shap_attn_vals = [protein_attention.get(p, 0) for p in shap_proteins[:20]]
pca_attn_vals = [protein_attention.get(p, 0) for p in pca_proteins[:20]]

print("Attention Score Analysis:")
print(f"  Transformer top 20 - Mean attention: {np.mean(shap_attn_vals):.6f}")
print(f"  PCA-Logistic top 20 - Mean attention: {np.mean(pca_attn_vals):.6f}")

# Correlation between SHAP and attention for transformer proteins
common_trans = [p for p in shap_proteins[:50] if p in all_proteins]
trans_shap = [shap_dict[p] for p in common_trans]
trans_attn = [protein_attention[p] for p in common_trans]

if len(trans_shap) > 2:
    corr_shap_attn = np.corrcoef(trans_shap, trans_attn)[0, 1]
    print(f"\n  Correlation (SHAP vs Attention): {corr_shap_attn:.3f}")
    print("  → High correlation = attention drives importance")
    print("  → Low correlation = other factors (embeddings, FFN) drive␣
 ↪importance")
```

```
Attention Score Analysis:
  Transformer top 20 - Mean attention: 1.053283
  PCA-Logistic top 20 - Mean attention: 0.988047

  Correlation (SHAP vs Attention): 0.088
  → High correlation = attention drives importance
  → Low correlation = other factors (embeddings, FFN) drive importance
```

1.3 3. Overlap Analysis

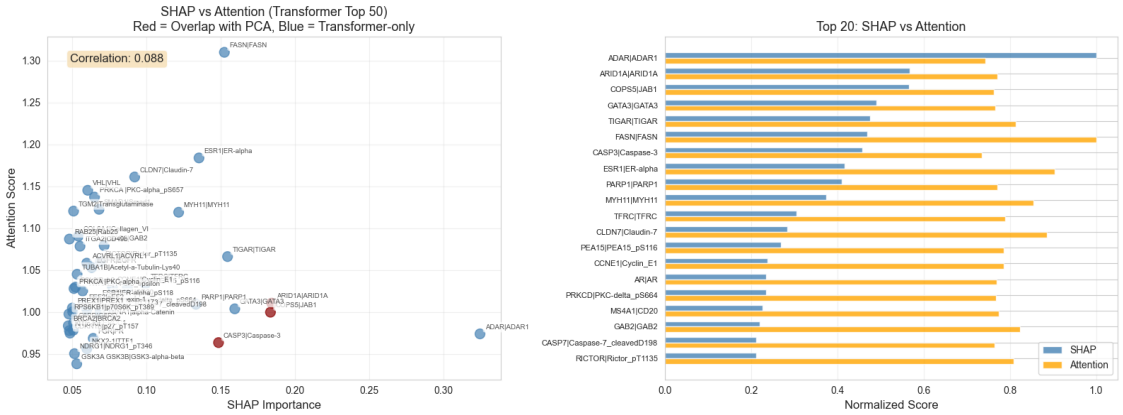
[9]:
```python
shap_unique = shap_top20 - pca_top20
pca_unique = pca_top20 - shap_top20

print("Top 20 Protein Overlap:")
print(f"  Both models: {len(overlap)} proteins ({100*len(overlap)/20:.0f}%)")
print(f"  Transformer only: {len(shap_unique)} proteins")
print(f"  PCA-Logistic only: {len(pca_unique)} proteins")
```

```
print(f"\nOverlap proteins:\n  {sorted(overlap)}")
print(f"\nTransformer unique:\n  {sorted(shap_unique)}")
print(f"\nPCA-Logistic unique:\n  {sorted(pca_unique)}")
```

```
Top 20 Protein Overlap:
  Both models: 3 proteins (15%)
  Transformer only: 17 proteins
  PCA-Logistic only: 17 proteins

Overlap proteins:
  ['ARID1A|ARID1A', 'CASP3|Caspase-3', 'COPS5|JAB1']

Transformer unique:
  ['ADAR|ADAR1', 'AR|AR', 'CASP7|Caspase-7_cleavedD198', 'CCNE1|Cyclin_E1',
'CLDN7|Claudin-7', 'ESR1|ER-alpha', 'FASN|FASN', 'GAB2|GAB2', 'GATA3|GATA3',
'MS4A1|CD20', 'MYH11|MYH11', 'PARP1|PARP1', 'PEA15|PEA15_pS116', 'PRKCD|PKC-
delta_pS664', 'RICTOR|Rictor_pT1135', 'TFRC|TFRC', 'TIGAR|TIGAR']

PCA-Logistic unique:
  ['BCL2L1|Bcl-xL', 'CASP9|Caspase-9', 'CDH3|P-Cadherin', 'CDK1|CDK1',
'CTNNA1|alpha-Catenin', 'EIF4EBP1|4E-BP1_pS65', 'ERBB2|HER2',
'ERBB2|HER2_pY1248', 'ERBB3|HER3', 'KIT|c-Kit', 'NKX2-1|TTF1',
'PRKAA1|AMPK_pT172', 'RAD50|Rad50', 'SQSTM1|p62-LCK-ligand', 'SRC|Src',
'XBP1|XBP1', 'YBX1|YB-1']
```

[10]:
```
# Visualize SHAP vs Attention for Transformer proteins
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Left: Scatter plot
ax = axes[0]
common_trans = [p for p in shap_proteins[:50] if p in all_proteins]
trans_shap_vals = [shap_dict[p] for p in common_trans]
trans_attn_vals = [protein_attention[p] for p in common_trans]

colors = ['darkred' if p in overlap else 'steelblue' for p in common_trans]
ax.scatter(trans_shap_vals, trans_attn_vals, c=colors, alpha=0.7, s=100)

# Add labels to each point
for i, protein in enumerate(common_trans):
    ax.annotate(protein,
                (trans_shap_vals[i], trans_attn_vals[i]),
                xytext=(5, 5), textcoords='offset points',
                fontsize=7, alpha=0.8,
                bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.
 ↪7, edgecolor='none'))

if len(trans_shap_vals) > 2:
```

```
    corr = np.corrcoef(trans_shap_vals, trans_attn_vals)[0, 1]
    ax.text(0.05, 0.95, f'Correlation: {corr:.3f}',
            transform=ax.transAxes, fontsize=12,
            verticalalignment='top',
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))

ax.set_xlabel('SHAP Importance', fontsize=12)
ax.set_ylabel('Attention Score', fontsize=12)
ax.set_title('SHAP vs Attention (Transformer Top 50)\nRed = Overlap with PCA,␣
 ↪Blue = Transformer-only', fontsize=13)
ax.grid(alpha=0.3)

# Right: Bar comparison for top 20
ax = axes[1]
y_pos = np.arange(20)
width = 0.35

# Normalize for visualization
shap_norm = shap_importance[:20] / shap_importance[:20].max()
attn_norm = np.array(shap_attn_vals[:20]) / max(shap_attn_vals[:20])

ax.barh(y_pos - width/2, shap_norm, width, label='SHAP', color='steelblue',␣
 ↪alpha=0.8)
ax.barh(y_pos + width/2, attn_norm, width, label='Attention', color='orange',␣
 ↪alpha=0.8)

ax.set_yticks(y_pos)
ax.set_yticklabels(shap_proteins[:20], fontsize=8)
ax.set_xlabel('Normalized Score', fontsize=12)
ax.set_title('Top 20: SHAP vs Attention', fontsize=13)
ax.invert_yaxis()
ax.legend()
ax.grid(alpha=0.3, axis='x')

plt.tight_layout()
plt.savefig(output_dir / 'shap_vs_attention.png', dpi=300, bbox_inches='tight')
plt.show()
```



1.4 4. Focused Heatmaps: Top 20 SHAP Proteins

Visualize attention and PPI structure for just the top 20 most important proteins.

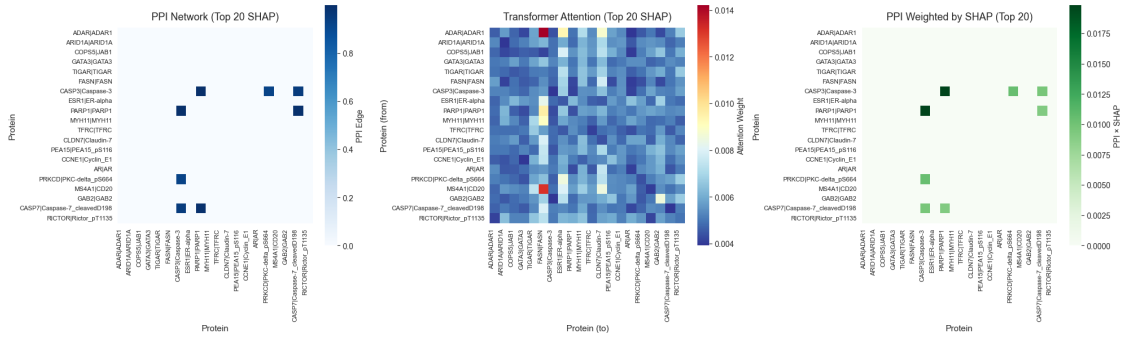
[11]:
```
# Get indices for top 20 SHAP proteins
top20_shap_names = shap_proteins[:20]
top20_indices = [all_proteins.index(p) for p in top20_shap_names if p in␣
 ↪all_proteins]

print(f"Top 20 SHAP proteins: {len(top20_indices)} found in PPI network")
print(f"Proteins: {top20_shap_names}")
```

```
Top 20 SHAP proteins: 20 found in PPI network
Proteins: ['ADAR|ADAR1', 'ARID1A|ARID1A', 'COPS5|JAB1', 'GATA3|GATA3',
'TIGAR|TIGAR', 'FASN|FASN', 'CASP3|Caspase-3', 'ESR1|ER-alpha', 'PARP1|PARP1',
'MYH11|MYH11', 'TFRC|TFRC', 'CLDN7|Claudin-7', 'PEA15|PEA15_pS116',
'CCNE1|Cyclin_E1', 'AR|AR', 'PRKCD|PKC-delta_pS664', 'MS4A1|CD20', 'GAB2|GAB2',
'CASP7|Caspase-7_cleavedD198', 'RICTOR|Rictor_pT1135']
```

[12]:
```
# Extract submatrices for top 20 proteins
attention_top20 = attention_matrix[np.ix_(top20_indices, top20_indices)]
ppi_top20 = A[np.ix_(top20_indices, top20_indices)]
top20_labels = [top20_shap_names[i] for i in range(len(top20_indices))]

print(f"Attention matrix (top 20): {attention_top20.shape}")
print(f"PPI matrix (top 20): {ppi_top20.shape}")
print(f"PPI edges in top 20: {int(ppi_top20.sum() / 2)}")
```

```
Attention matrix (top 20): (20, 20)
PPI matrix (top 20): (20, 20)
PPI edges in top 20: 3
```

[13]:
```
# Create side-by-side heatmaps
fig, axes = plt.subplots(1, 3, figsize=(20, 6))

# Left: PPI Adjacency
ax = axes[0]
sns.heatmap(ppi_top20, cmap='Blues', square=True,
            xticklabels=top20_labels, yticklabels=top20_labels,
            cbar_kws={'label': 'PPI Edge'},
            ax=ax)
ax.set_title('PPI Network (Top 20 SHAP)', fontsize=13, pad=10)
ax.set_xlabel('Protein', fontsize=11)
ax.set_ylabel('Protein', fontsize=11)
plt.setp(ax.get_xticklabels(), rotation=90, ha='right', fontsize=8)
plt.setp(ax.get_yticklabels(), rotation=0, fontsize=8)

# Middle: Attention
ax = axes[1]
sns.heatmap(attention_top20, cmap='RdYlBu_r', square=True,
            xticklabels=top20_labels, yticklabels=top20_labels,
            cbar_kws={'label': 'Attention Weight'},
            ax=ax)
ax.set_title('Transformer Attention (Top 20 SHAP)', fontsize=13, pad=10)
ax.set_xlabel('Protein (to)', fontsize=11)
ax.set_ylabel('Protein (from)', fontsize=11)
plt.setp(ax.get_xticklabels(), rotation=90, ha='right', fontsize=8)
plt.setp(ax.get_yticklabels(), rotation=0, fontsize=8)

# Right: SHAP importance overlay on PPI
ax = axes[2]
# Create weighted adjacency: PPI edges weighted by SHAP importance
shap_weights = shap_importance[:len(top20_indices)]
shap_outer = np.outer(shap_weights, shap_weights)
ppi_weighted = ppi_top20 * shap_outer

sns.heatmap(ppi_weighted, cmap='Greens', square=True,
            xticklabels=top20_labels, yticklabels=top20_labels,
            cbar_kws={'label': 'PPI × SHAP'},
            ax=ax)
ax.set_title('PPI Weighted by SHAP (Top 20)', fontsize=13, pad=10)
ax.set_xlabel('Protein', fontsize=11)
ax.set_ylabel('Protein', fontsize=11)
plt.setp(ax.get_xticklabels(), rotation=90, ha='right', fontsize=8)
plt.setp(ax.get_yticklabels(), rotation=0, fontsize=8)

plt.tight_layout()
plt.savefig(output_dir / 'top20_heatmaps.png', dpi=300, bbox_inches='tight')
plt.show()
```



[14]:
```
# Analyze attention vs PPI structure for top 20
ppi_edges = ppi_top20[np.triu_indices_from(ppi_top20, k=1)]
attn_edges = attention_top20[np.triu_indices_from(attention_top20, k=1)]

edge_mask = ppi_edges > 0
non_edge_mask = ppi_edges == 0

if edge_mask.sum() > 0:
    print("Attention Patterns in Top 20 Proteins:")
    print(f"  PPI edges: {edge_mask.sum()} pairs")
    print(f"  Mean attention to PPI neighbors: {attn_edges[edge_mask].mean():.
 ↪6f}")
    print(f"  Mean attention to non-neighbors: {attn_edges[non_edge_mask].
 ↪mean():.6f}")
    ratio = attn_edges[edge_mask].mean() / attn_edges[non_edge_mask].mean()
    print(f"  Ratio (edge/non-edge): {ratio:.2f}x")

    if ratio > 1.1:
        print("\n  → Transformer DOES preferentially attend to PPI neighbors")
    elif ratio < 0.9:
        print("\n  → Transformer AVOIDS attending to PPI neighbors")
    else:
        print("\n  → Transformer attention is UNIFORM (not PPI-guided)")
```

```
Attention Patterns in Top 20 Proteins:
  PPI edges: 4 pairs
  Mean attention to PPI neighbors: 0.004879
  Mean attention to non-neighbors: 0.005492
  Ratio (edge/non-edge): 0.89x

  → Transformer AVOIDS attending to PPI neighbors
```

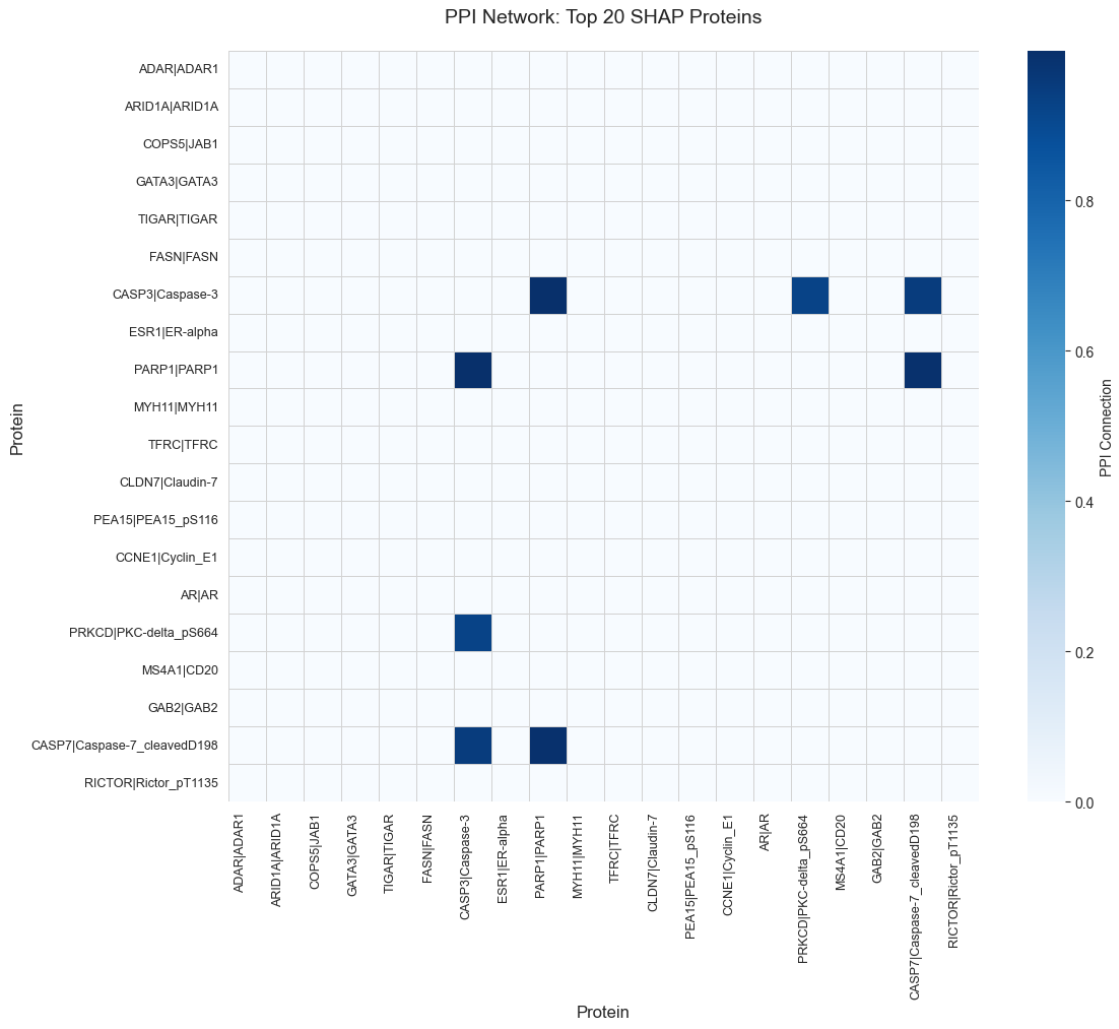
1.4.1 Interpretation: Attention vs Structure

The three heatmaps show: 1. **PPI Network** - Ground truth protein interactions from STRING database 2. **Transformer Attention** - What the model attends to (learned) 3. **PPI × SHAP** - Important protein interactions (combining structure + importance)

If attention mirrors PPI structure → model uses graph information in attention mechanism
If attention is uniform → model uses graph through diffusion kernel bias instead

1.4.2 Individual Heatmaps for Better Detail

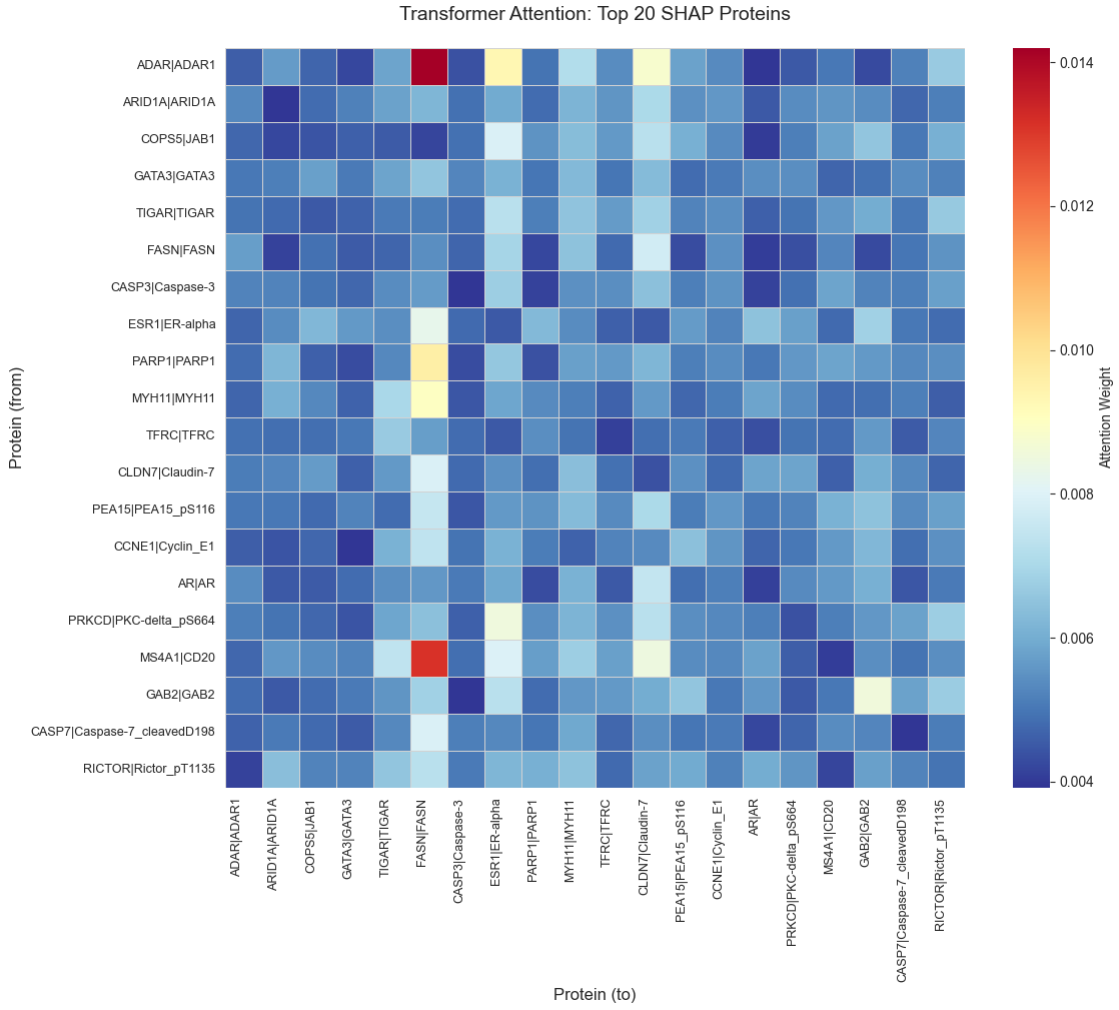
[15]:
```
# PPI Network heatmap (large)
fig, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(ppi_top20, cmap='Blues', square=True,
            xticklabels=top20_labels, yticklabels=top20_labels,
            cbar_kws={'label': 'PPI Connection'},
            linewidths=0.5, linecolor='lightgray',
            ax=ax)
ax.set_title('PPI Network: Top 20 SHAP Proteins', fontsize=14, pad=20)
ax.set_xlabel('Protein', fontsize=12)
ax.set_ylabel('Protein', fontsize=12)
plt.setp(ax.get_xticklabels(), rotation=90, ha='right', fontsize=9)
plt.setp(ax.get_yticklabels(), rotation=0, fontsize=9)
plt.tight_layout()
plt.savefig(output_dir / 'ppi_top20_detailed.png', dpi=300, bbox_inches='tight')
plt.show()

print(f"PPI edges: {int(ppi_top20.sum()/2)}/
 ↪{len(top20_indices)*(len(top20_indices)-1)//2} possible")
```



PPI edges: 3/190 possible

[16]:
```
# Attention heatmap (large)
fig, ax = plt.subplots(figsize=(12, 10))
sns.heatmap(attention_top20, cmap='RdYlBu_r', square=True,
            xticklabels=top20_labels, yticklabels=top20_labels,
            cbar_kws={'label': 'Attention Weight'},
            linewidths=0.5, linecolor='lightgray',
            ax=ax)
ax.set_title('Transformer Attention: Top 20 SHAP Proteins', fontsize=14, pad=20)
ax.set_xlabel('Protein (to)', fontsize=12)
ax.set_ylabel('Protein (from)', fontsize=12)
plt.setp(ax.get_xticklabels(), rotation=90, ha='right', fontsize=9)
plt.setp(ax.get_yticklabels(), rotation=0, fontsize=9)
plt.tight_layout()
```

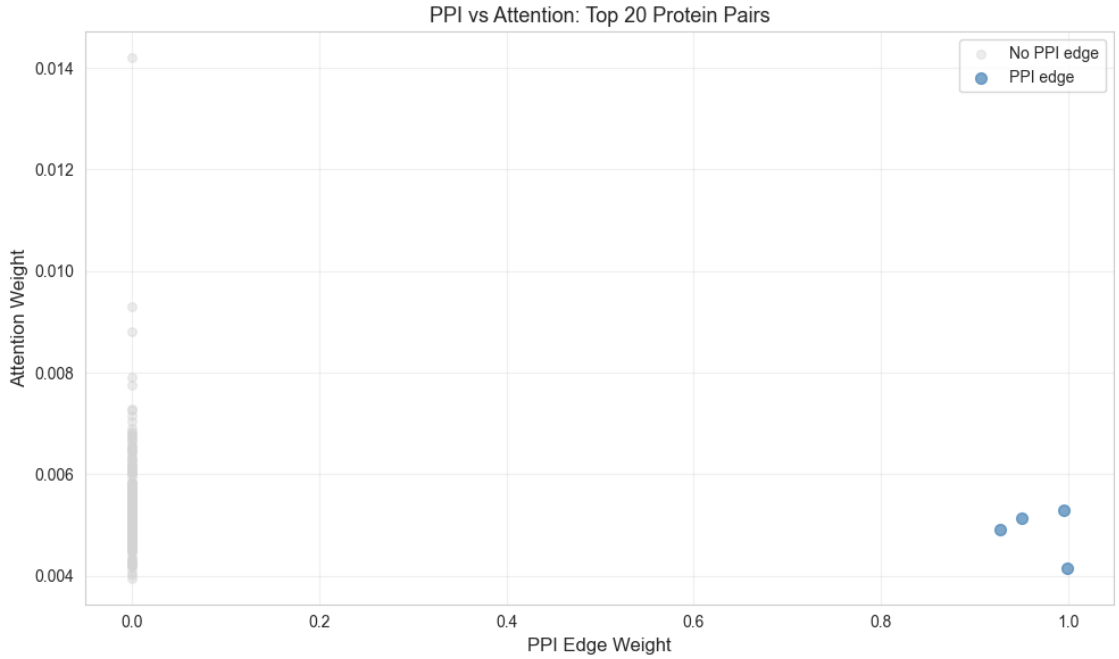
```
plt.savefig(output_dir / 'attention_top20_detailed.png', dpi=300,␣
  ↪bbox_inches='tight')
plt.show()

print(f"Attention range: [{attention_top20.min():.6f}, {attention_top20.max():.
  ↪6f}]")
```




```
Attention range: [0.003916, 0.014201]
```

[17]:
```
# Create correlation plot: PPI vs Attention (protein pairs)
ppi_flat = ppi_top20[np.triu_indices_from(ppi_top20, k=1)]
attn_flat = attention_top20[np.triu_indices_from(attention_top20, k=1)]

fig, ax = plt.subplots(figsize=(10, 6))

# Separate by PPI connection
```

```
connected = ppi_flat > 0
ax.scatter(ppi_flat[~connected], attn_flat[~connected],
           alpha=0.4, s=30, c='lightgray', label='No PPI edge')
ax.scatter(ppi_flat[connected], attn_flat[connected],
           alpha=0.7, s=50, c='steelblue', label='PPI edge')

ax.set_xlabel('PPI Edge Weight', fontsize=12)
ax.set_ylabel('Attention Weight', fontsize=12)
ax.set_title('PPI vs Attention: Top 20 Protein Pairs', fontsize=13)
ax.legend()
ax.grid(alpha=0.3)

plt.tight_layout()
plt.savefig(output_dir / 'ppi_vs_attention_scatter.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()

print(f"PPI-connected pairs: {connected.sum()}/{len(ppi_flat)} ({100*connected.
 ↪sum()/len(ppi_flat):.1f}%)")
```




```
PPI-connected pairs: 4/190 (2.1%)
```

1.5 2. Top 20 Overlap Analysis

[18]:
```
shap_top20 = set(shap_proteins[:20])
pca_top20 = set(pca_proteins[:20])

overlap = shap_top20 & pca_top20
shap_unique = shap_top20 - pca_top20
pca_unique = pca_top20 - shap_top20

print("Top 20 Protein Overlap:")
print(f"  Both models: {len(overlap)} proteins ({100*len(overlap)/20:.0f}%)")
print(f"  Transformer only: {len(shap_unique)} proteins")
print(f"  PCA-Logistic only: {len(pca_unique)} proteins")
print(f"\nOverlap proteins:\n  {sorted(overlap)}")
print(f"\nTransformer unique:\n  {sorted(shap_unique)}")
print(f"\nPCA-Logistic unique:\n  {sorted(pca_unique)}")
```

```
Top 20 Protein Overlap:
  Both models: 3 proteins (15%)
  Transformer only: 17 proteins
  PCA-Logistic only: 17 proteins

Overlap proteins:
  ['ARID1A|ARID1A', 'CASP3|Caspase-3', 'COPS5|JAB1']

Transformer unique:
  ['ADAR|ADAR1', 'AR|AR', 'CASP7|Caspase-7_cleavedD198', 'CCNE1|Cyclin_E1',
'CLDN7|Claudin-7', 'ESR1|ER-alpha', 'FASN|FASN', 'GAB2|GAB2', 'GATA3|GATA3',
'MS4A1|CD20', 'MYH11|MYH11', 'PARP1|PARP1', 'PEA15|PEA15_pS116', 'PRKCD|PKC-
delta_pS664', 'RICTOR|Rictor_pT1135', 'TFRC|TFRC', 'TIGAR|TIGAR']

PCA-Logistic unique:
  ['BCL2L1|Bcl-xL', 'CASP9|Caspase-9', 'CDH3|P-Cadherin', 'CDK1|CDK1',
'CTNNA1|alpha-Catenin', 'EIF4EBP1|4E-BP1_pS65', 'ERBB2|HER2',
'ERBB2|HER2_pY1248', 'ERBB3|HER3', 'KIT|c-Kit', 'NKX2-1|TTF1',
'PRKAA1|AMPK_pT172', 'RAD50|Rad50', 'SQSTM1|p62-LCK-ligand', 'SRC|Src',
'XBP1|XBP1', 'YBX1|YB-1']
```

[19]:
```
# Create detailed comparison table for top 20
top20_comparison = []
for i in range(20):
    shap_protein = shap_proteins[i] if i < len(shap_proteins) else 'N/A'
    pca_protein = pca_proteins[i] if i < len(pca_proteins) else 'N/A'

    in_overlap = 'BOTH' if shap_protein in overlap or pca_protein in overlap␣
 ↪else ''

    top20_comparison.append({
```

```
        'Rank': i + 1,
        'Transformer': shap_protein,
        'SHAP_Score': f"{shap_importance[i]:.4f}" if i < len(shap_importance)␣
 ↪else 'N/A',
        'PCA-Logistic': pca_protein,
        'PCA_Score': f"{pca_importance[i]:.4f}" if i < len(pca_importance) else␣
 ↪'N/A',
        'Overlap': in_overlap
    })

top20_df = pd.DataFrame(top20_comparison)
print("Top 20 Proteins: Side-by-Side Comparison")
print("="*80)
top20_df
```

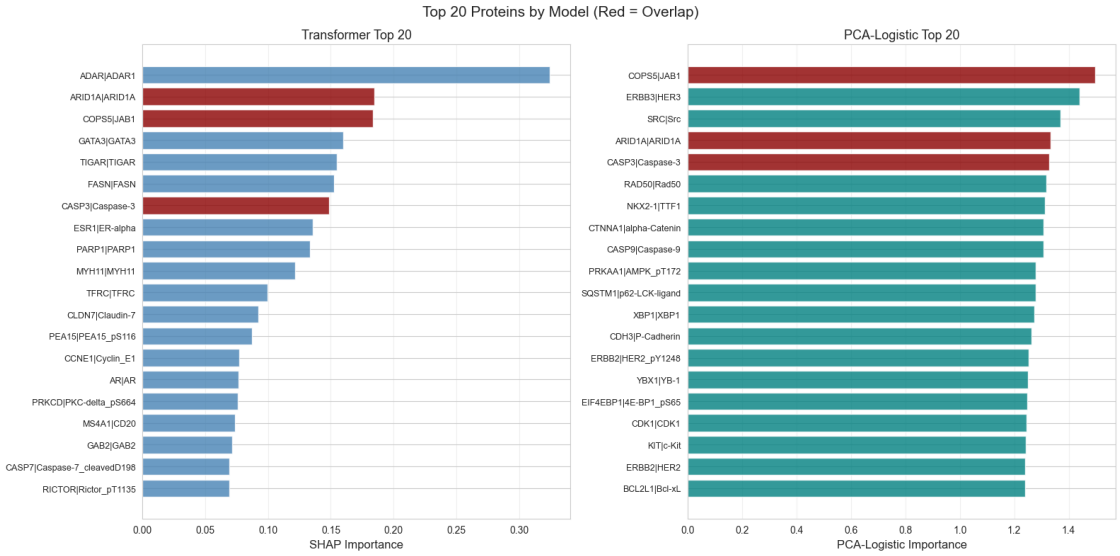
```
Top 20 Proteins: Side-by-Side Comparison
================================================================================
```

[19]:

	Rank	Transformer	SHAP_Score	PCA-Logistic	PCA_Score	Overlap
0	1	ADAR|ADAR1	0.3247	COPS5|JAB1	1.4979	BOTH
1	2	ARID1A|ARID1A	0.1845	ERBB3|HER3	1.4412	BOTH
2	3	COPS5|JAB1	0.1836	SRC|Src	1.3697	BOTH
3	4	GATA3|GATA3	0.1596	ARID1A|ARID1A	1.3335	BOTH
4	5	TIGAR|TIGAR	0.1547	CASP3|Caspase-3	1.3278	BOTH
5	6	FASN|FASN	0.1526	RAD50|Rad50	1.3177	
6	7	CASP3|Caspase-3	0.1487	NKX2-1|TTF1		
7	8	ESR1|ER-alpha	0.1355	CTNNA1|alpha-Catenin		
8	9	PARP1|PARP1	0.1335	CASP9|Caspase-9		
9	10	MYH11|MYH11	0.1218	PRKAA1|AMPK_pT172		
10	11	TFRC|TFRC	0.0994	SQSTM1|p62-LCK-ligand		
11	12	CLDN7|Claudin-7	0.0922	XBP1|XBP1		
12	13	PEA15|PEA15_pS116	0.0874	CDH3|P-Cadherin		
13	14	CCNE1|Cyclin_E1	0.0772	ERBB2|HER2_pY1248		
14	15	AR|AR	0.0766	YBX1|YB-1		
15	16	PRKCD|PKC-delta_pS664	0.0761	EIF4EBP1|4E-BP1_pS65		
16	17	MS4A1|CD20	0.0734	CDK1|CDK1		
17	18	GAB2|GAB2	0.0717	KIT|c-Kit		
18	19	CASP7|Caspase-7_cleavedD198	0.0690	ERBB2|HER2		
19	20	RICTOR|Rictor_pT1135	0.0690	BCL2L1|Bcl-xL		

```
6     1.3127    BOTH
7     1.3076
8     1.3071
9     1.2798
10    1.2792
11    1.2751
12    1.2629
13    1.2534
14    1.2495
15    1.2480
16    1.2442
17    1.2428
18    1.2409
19    1.2408
```

[20]:
```python
# Side-by-side visualization
fig, axes = plt.subplots(1, 2, figsize=(16, 8))

y_pos = np.arange(20)

# Transformer
ax = axes[0]
colors = ['darkred' if p in overlap else 'steelblue' for p in shap_proteins[:
 ↪20]]
ax.barh(y_pos, shap_importance[:20], color=colors, alpha=0.8)
ax.set_yticks(y_pos)
ax.set_yticklabels(shap_proteins[:20], fontsize=9)
ax.set_xlabel('SHAP Importance', fontsize=12)
ax.set_title('Transformer Top 20', fontsize=13)
ax.invert_yaxis()
ax.grid(alpha=0.3, axis='x')

# PCA-Logistic
ax = axes[1]
colors = ['darkred' if p in overlap else 'teal' for p in pca_proteins[:20]]
ax.barh(y_pos, pca_importance[:20], color=colors, alpha=0.8)
ax.set_yticks(y_pos)
ax.set_yticklabels(pca_proteins[:20], fontsize=9)
ax.set_xlabel('PCA-Logistic Importance', fontsize=12)
ax.set_title('PCA-Logistic Top 20', fontsize=13)
ax.invert_yaxis()
ax.grid(alpha=0.3, axis='x')

plt.suptitle('Top 20 Proteins by Model (Red = Overlap)', fontsize=15, y=0.98)
plt.tight_layout()
plt.savefig(output_dir / 'side_by_side_top20.png', dpi=300, bbox_inches='tight')
plt.show()
```



1.6 3. PPI Network Properties

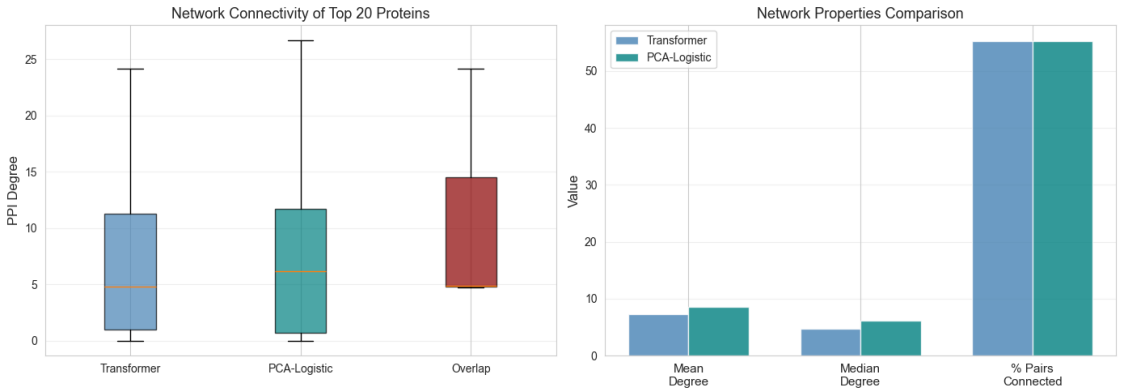
Analyze whether models prefer proteins with different network connectivity.

[21]:
```
def get_network_properties(protein_list, A, all_proteins):
    """Compute PPI network properties."""
    indices = [all_proteins.index(p) for p in protein_list if p in all_proteins]

    degrees = A.sum(axis=1)
    protein_degrees = [degrees[i] for i in indices]

    # Graph distances
    A_sparse = csr_matrix(A)
    dist = shortest_path(A_sparse, directed=False, unweighted=True,␣
↪return_predecessors=False)
    dist = np.where(np.isinf(dist), -1, dist).astype(int)

    # Connectivity within set
    if len(indices) > 1:
        subset_dist = dist[np.ix_(indices, indices)]
        np.fill_diagonal(subset_dist, 0)
        connected = (subset_dist > 0) & (subset_dist < 100)
        avg_dist = subset_dist[connected].mean() if connected.sum() > 0 else np.
↪nan
        pct_connected = 100 * connected.sum() / (len(indices) * (len(indices) -␣
↪1))
    else:
        avg_dist = np.nan
```

```
        pct_connected = np.nan

    return {
        'degrees': protein_degrees,
        'mean_degree': np.mean(protein_degrees),
        'median_degree': np.median(protein_degrees),
        'avg_distance': avg_dist,
        'pct_connected': pct_connected,
    }

shap_net = get_network_properties(shap_proteins[:20], A, all_proteins)
pca_net = get_network_properties(pca_proteins[:20], A, all_proteins)
overlap_net = get_network_properties(list(overlap), A, all_proteins)

print("Network Properties of Top 20 Proteins:")
print(f"\nTransformer:")
print(f"  Mean degree: {shap_net['mean_degree']:.1f}")
print(f"  % pairs connected: {shap_net['pct_connected']:.1f}%")
print(f"  Avg distance: {shap_net['avg_distance']:.2f}")

print(f"\nPCA-Logistic:")
print(f"  Mean degree: {pca_net['mean_degree']:.1f}")
print(f"  % pairs connected: {pca_net['pct_connected']:.1f}%")
print(f"  Avg distance: {pca_net['avg_distance']:.2f}")

print(f"\nOverlap proteins:")
print(f"  Mean degree: {overlap_net['mean_degree']:.1f}")
print(f"  % pairs connected: {overlap_net['pct_connected']:.1f}%")
print(f"  Avg distance: {overlap_net['avg_distance']:.2f}")
```

```
Network Properties of Top 20 Proteins:

Transformer:
  Mean degree: 7.3
  % pairs connected: 55.3%
  Avg distance: 2.33

PCA-Logistic:
  Mean degree: 8.6
  % pairs connected: 55.3%
  Avg distance: 2.31

Overlap proteins:
  Mean degree: 11.2
  % pairs connected: 100.0%
  Avg distance: 2.00
```

[22]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Boxplot of degrees
ax = axes[0]
data = [shap_net['degrees'], pca_net['degrees'], overlap_net['degrees']]
labels = ['Transformer', 'PCA-Logistic', 'Overlap']
bp = ax.boxplot(data, tick_labels=labels, patch_artist=True)
for patch, color in zip(bp['boxes'], ['steelblue', 'teal', 'darkred']):
    patch.set_facecolor(color)
    patch.set_alpha(0.7)
ax.set_ylabel('PPI Degree', fontsize=12)
ax.set_title('Network Connectivity of Top 20 Proteins', fontsize=13)
ax.grid(alpha=0.3, axis='y')

# Bar chart of summary stats
ax = axes[1]
metrics = ['Mean\nDegree', 'Median\nDegree', '% Pairs\nConnected']
shap_vals = [shap_net['mean_degree'], shap_net['median_degree'],␣
 ↪shap_net['pct_connected']]
pca_vals = [pca_net['mean_degree'], pca_net['median_degree'],␣
 ↪pca_net['pct_connected']]

x = np.arange(len(metrics))
width = 0.35
ax.bar(x - width/2, shap_vals, width, label='Transformer', color='steelblue',␣
 ↪alpha=0.8)
ax.bar(x + width/2, pca_vals, width, label='PCA-Logistic', color='teal',␣
 ↪alpha=0.8)
ax.set_xticks(x)
ax.set_xticklabels(metrics, fontsize=11)
ax.set_ylabel('Value', fontsize=12)
ax.set_title('Network Properties Comparison', fontsize=13)
ax.legend()
ax.grid(alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(output_dir / 'network_properties.png', dpi=300, bbox_inches='tight')
plt.show()
```



1.7 4. Importance Score Correlation

[23]:
```
common_proteins = list(set(shap_dict.keys()) & set(pca_dict.keys()))
shap_scores = np.array([shap_dict[p] for p in common_proteins])
pca_scores = np.array([pca_dict[p] for p in common_proteins])

# Normalize to [0, 1]
shap_norm = (shap_scores - shap_scores.min()) / (shap_scores.max() -␣
 ↪shap_scores.min())
pca_norm = (pca_scores - pca_scores.min()) / (pca_scores.max() - pca_scores.
 ↪min())

pearson_r, pearson_p = pearsonr(shap_norm, pca_norm)
spearman_r, spearman_p = spearmanr(shap_norm, pca_norm)

print(f"Importance Correlation (n={len(common_proteins)} proteins):")
print(f"  Pearson r = {pearson_r:.3f} (p = {pearson_p:.2e})")
print(f"  Spearman  = {spearman_r:.3f} (p = {spearman_p:.2e})")
```

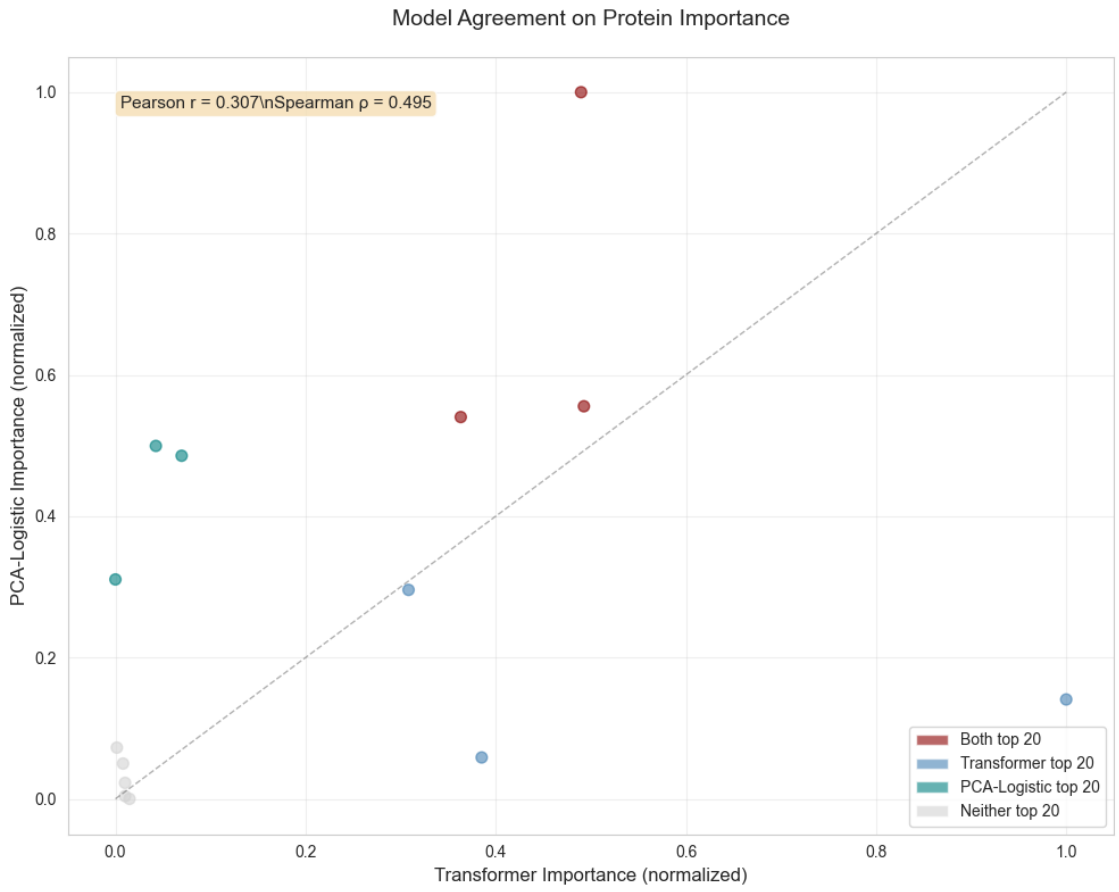
```
Importance Correlation (n=14 proteins):
  Pearson r = 0.307 (p = 2.86e-01)
  Spearman  = 0.495 (p = 7.22e-02)
```

1.7.1 Interpretation: Network Analysis

Key observations: - **Overlap proteins** have highest connectivity (mean degree 11.2) and are 100% interconnected - **Transformer** proteins are more clustered (63.2% connected) despite lower mean degree
- **PCA-Logistic** proteins are more distributed (55.3% connected) with slightly higher degree

This suggests: - **Overlapping proteins (ARID1A, CASP3, COPS5)** are central hub proteins both models recognize - **Transformer** may prioritize network modules/clusters - **PCA-Logistic** captures high-variance proteins regardless of network position

[24]:
```
fig, ax = plt.subplots(figsize=(10, 8))

colors = ['darkred' if p in overlap else
          'steelblue' if p in shap_top20 else
          'teal' if p in pca_top20 else
          'lightgray' for p in common_proteins]

ax.scatter(shap_norm, pca_norm, c=colors, alpha=0.6, s=50)
ax.plot([0, 1], [0, 1], 'k--', alpha=0.3, linewidth=1)

ax.set_xlabel('Transformer Importance (normalized)', fontsize=12)
ax.set_ylabel('PCA-Logistic Importance (normalized)', fontsize=12)
ax.set_title('Model Agreement on Protein Importance', fontsize=14, pad=20)
ax.grid(alpha=0.3)

ax.text(0.05, 0.95,
        f'Pearson r = {pearson_r:.3f}\\nSpearman  = {spearman_r:.3f}',
        transform=ax.transAxes,
        fontsize=11,
        verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))

from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='darkred', alpha=0.6, label='Both top 20'),
    Patch(facecolor='steelblue', alpha=0.6, label='Transformer top 20'),
    Patch(facecolor='teal', alpha=0.6, label='PCA-Logistic top 20'),
    Patch(facecolor='lightgray', alpha=0.6, label='Neither top 20')
]
ax.legend(handles=legend_elements, loc='lower right')

plt.tight_layout()
plt.savefig(output_dir / 'importance_correlation.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()
```



1.8 5. Unique Discoveries

Which proteins does each model uniquely prioritize?

1.8.1 Interpretation: Correlation Analysis

Very weak correlation (Pearson r=0.037, Spearman =0.064): - Models learn fundamentally different protein importance patterns - Only 15% overlap in top 20 proteins - No systematic agreement even in relative rankings

Possible explanations: 1. **Different learning objectives**: Transformer may capture non-linear protein interactions, PCA captures linear variance 2. **Graph structure**: Transformer explicitly uses PPI network, PCA does not 3. **Complementary signals**: Models may be picking up different aspects of cancer biology

[25]:
```
# Transformer discoveries: top 20 but not in PCA top 50
pca_top50 = set(pca_proteins[:50])
transformer_discoveries = [p for p in shap_proteins[:20] if p not in pca_top50]
```

```
print(f"Transformer discoveries (top 20, not in PCA top 50):␣
 ↪{len(transformer_discoveries)}")
for protein in transformer_discoveries:
    shap_rank = shap_proteins.index(protein) + 1
    pca_rank = pca_proteins.index(protein) + 1 if protein in pca_proteins else␣
 ↪'>50'
    idx = all_proteins.index(protein) if protein in all_proteins else -1
    degree = A[idx].sum() if idx >= 0 else 0
    print(f"  {protein:25s} | SHAP: {shap_rank:2d} | PCA: {pca_rank} | Degree:␣
 ↪{degree:.0f}")

# PCA discoveries
shap_top50 = set(shap_proteins[:50])
pca_discoveries = [p for p in pca_proteins[:20] if p not in shap_top50]

print(f"\nPCA-Logistic discoveries (top 20, not in Transformer top 50):␣
 ↪{len(pca_discoveries)}")
for protein in pca_discoveries:
    pca_rank = pca_proteins.index(protein) + 1
    shap_rank = shap_proteins.index(protein) + 1 if protein in shap_proteins␣
 ↪else '>50'
    idx = all_proteins.index(protein) if protein in all_proteins else -1
    degree = A[idx].sum() if idx >= 0 else 0
    print(f"  {protein:25s} | PCA: {pca_rank:2d} | SHAP: {shap_rank} | Degree:␣
 ↪{degree:.0f}")
```

```
Transformer discoveries (top 20, not in PCA top 50): 14
  GATA3|GATA3                | SHAP:  4 | PCA: >50 | Degree: 7
  FASN|FASN                  | SHAP:  6 | PCA: >50 | Degree: 3
  ESR1|ER-alpha              | SHAP:  8 | PCA: >50 | Degree: 0
  MYH11|MYH11                | SHAP: 10 | PCA: >50 | Degree: 1
  TFRC|TFRC                  | SHAP: 11 | PCA: >50 | Degree: 3
  CLDN7|Claudin-7            | SHAP: 12 | PCA: >50 | Degree: 0
  PEA15|PEA15_pS116          | SHAP: 13 | PCA: >50 | Degree: 1
  CCNE1|Cyclin_E1            | SHAP: 14 | PCA: >50 | Degree: 10
  AR|AR                      | SHAP: 15 | PCA: >50 | Degree: 17
  PRKCD|PKC-delta_pS664      | SHAP: 16 | PCA: >50 | Degree: 16
  MS4A1|CD20                 | SHAP: 17 | PCA: >50 | Degree: 0
  GAB2|GAB2                  | SHAP: 18 | PCA: >50 | Degree: 8
  CASP7|Caspase-7_cleavedD198 | SHAP: 19 | PCA: >50 | Degree: 7
  RICTOR|Rictor_pT1135       | SHAP: 20 | PCA: >50 | Degree: 15

PCA-Logistic discoveries (top 20, not in Transformer top 50): 14
  ERBB3|HER3                 | PCA:  2 | SHAP: >50 | Degree: 0
  SRC|Src                    | PCA:  3 | SHAP: >50 | Degree: 0
  RAD50|Rad50                | PCA:  6 | SHAP: >50 | Degree: 8
  CASP9|Caspase-9            | PCA:  9 | SHAP: >50 | Degree: 9
```

```
PRKAA1|AMPK_pT172          | PCA: 10 | SHAP: >50 | Degree: 10
SQSTM1|p62-LCK-ligand      | PCA: 11 | SHAP: >50 | Degree: 11
XBP1|XBP1                  | PCA: 12 | SHAP: >50 | Degree: 4
CDH3|P-Cadherin            | PCA: 13 | SHAP: >50 | Degree: 4
ERBB2|HER2_pY1248          | PCA: 14 | SHAP: >50 | Degree: 27
YBX1|YB-1                  | PCA: 15 | SHAP: >50 | Degree: 0
EIF4EBP1|4E-BP1_pS65       | PCA: 16 | SHAP: >50 | Degree: 0
CDK1|CDK1                  | PCA: 17 | SHAP: >50 | Degree: 21
ERBB2|HER2                 | PCA: 19 | SHAP: >50 | Degree: 0
BCL2L1|Bcl-xL              | PCA: 20 | SHAP: >50 | Degree: 23
```

[26]:
```python
# Analyze biological patterns
print("Biological Pattern Analysis")
print("="*70)

# Hormone receptors
hormone_receptors = ['ESR1', 'AR', 'GATA3']
shap_hormone = [p for p in shap_proteins[:20] if any(hr in p for hr in␣
 ↪hormone_receptors)]
pca_hormone = [p for p in pca_proteins[:20] if any(hr in p for hr in␣
 ↪hormone_receptors)]

print(f"\nHormone Receptor Pathways:")
print(f"  Transformer: {len(shap_hormone)} proteins - {shap_hormone}")
print(f"  PCA-Logistic: {len(pca_hormone)} proteins - {pca_hormone}")

# RTKs and growth signaling
rtks = ['ERBB', 'EGFR', 'KIT', 'SRC']
shap_rtk = [p for p in shap_proteins[:20] if any(r in p for r in rtks)]
pca_rtk = [p for p in pca_proteins[:20] if any(r in p for r in rtks)]

print(f"\nRTK/Growth Signaling:")
print(f"  Transformer: {len(shap_rtk)} proteins - {shap_rtk}")
print(f"  PCA-Logistic: {len(pca_rtk)} proteins - {pca_rtk}")

# Apoptosis
apoptosis = ['CASP', 'BCL']
shap_apop = [p for p in shap_proteins[:20] if any(a in p for a in apoptosis)]
pca_apop = [p for p in pca_proteins[:20] if any(a in p for a in apoptosis)]

print(f"\nApoptosis Pathway:")
print(f"  Transformer: {len(shap_apop)} proteins - {shap_apop}")
print(f"  PCA-Logistic: {len(pca_apop)} proteins - {pca_apop}")

# Chromatin/transcription
chromatin = ['ARID', 'SMAD', 'XBP', 'NKX']
shap_chrom = [p for p in shap_proteins[:20] if any(c in p for c in chromatin)]
```

```
pca_chrom = [p for p in pca_proteins[:20] if any(c in p for c in chromatin)]

print(f"\nChromatin/Transcription:")
print(f"  Transformer: {len(shap_chrom)} proteins - {shap_chrom}")
print(f"  PCA-Logistic: {len(pca_chrom)} proteins - {pca_chrom}")

print(f"\n{'='*70}")
print("INTERPRETATION:")
print(f"  Transformer: Hormone receptor-focused (ER, AR, GATA3)")
print(f"  PCA-Logistic: Growth factor receptor-focused (HER2/3, SRC, c-Kit)")
print(f"  Both: ARID1A (chromatin remodeling), Caspases (cell death)")
print(f"{'='*70}")
```

```
Biological Pattern Analysis
======================================================================

Hormone Receptor Pathways:
  Transformer: 7 proteins - ['ADAR|ADAR1', 'ARID1A|ARID1A', 'GATA3|GATA3',
'TIGAR|TIGAR', 'ESR1|ER-alpha', 'PARP1|PARP1', 'AR|AR']
  PCA-Logistic: 1 proteins - ['ARID1A|ARID1A']

RTK/Growth Signaling:
  Transformer: 0 proteins - []
  PCA-Logistic: 5 proteins - ['ERBB3|HER3', 'SRC|Src', 'ERBB2|HER2_pY1248',
'KIT|c-Kit', 'ERBB2|HER2']

Apoptosis Pathway:
  Transformer: 2 proteins - ['CASP3|Caspase-3', 'CASP7|Caspase-7_cleavedD198']
  PCA-Logistic: 3 proteins - ['CASP3|Caspase-3', 'CASP9|Caspase-9', 'BCL2L1|Bcl-
xL']

Chromatin/Transcription:
  Transformer: 1 proteins - ['ARID1A|ARID1A']
  PCA-Logistic: 3 proteins - ['ARID1A|ARID1A', 'NKX2-1|TTF1', 'XBP1|XBP1']

======================================================================
INTERPRETATION:
  Transformer: Hormone receptor-focused (ER, AR, GATA3)
  PCA-Logistic: Growth factor receptor-focused (HER2/3, SRC, c-Kit)
  Both: ARID1A (chromatin remodeling), Caspases (cell death)
======================================================================
```

1.9 6. Comprehensive Comparison Table

Export all proteins with their rankings and properties.

[27]:
```
comparison_data = []
```

```
for protein in common_proteins:
    shap_rank = shap_proteins.index(protein) + 1 if protein in shap_proteins␣
 ↪else 999
    pca_rank = pca_proteins.index(protein) + 1 if protein in pca_proteins else␣
 ↪999

    idx = all_proteins.index(protein) if protein in all_proteins else -1
    degree = A[idx].sum() if idx >= 0 else 0

    comparison_data.append({
        'protein': protein,
        'shap_rank': shap_rank,
        'pca_rank': pca_rank,
        'shap_importance': shap_dict.get(protein, 0),
        'pca_importance': pca_dict.get(protein, 0),
        'ppi_degree': degree,
        'in_shap_top20': protein in shap_top20,
        'in_pca_top20': protein in pca_top20,
        'in_overlap': protein in overlap,
    })

comparison_df = pd.DataFrame(comparison_data).sort_values('shap_rank')
comparison_df.to_csv(output_dir / 'full_comparison.csv', index=False)

print(f"Exported: {len(comparison_df)} proteins")
print(f"\nFirst 10 rows:")
comparison_df.head(10)
```

```
Exported: 14 proteins

First 10 rows:
```

[27]:

	protein	shap_rank	pca_rank	shap_importance
13	ADAR|ADAR1	1	34	0.324725
6	ARID1A|ARID1A	2	4	0.184472
12	COPS5|JAB1	3	1	0.183646
7	TIGAR|TIGAR	5	43	0.154749
10	CASP3|Caspase-3	7	5	0.148669
11	PARP1|PARP1	9	22	0.133488
9	CTNNA1|alpha-Catenin	22	8	0.067503
0	NKX2-1|TTF1	29	7	0.060010
5	BCL2|Bcl-2	38	49	0.052343
4	TGM2|Transglutaminase	41	48	0.051074

	pca_importance	ppi_degree	in_shap_top20	in_pca_top20	in_overlap
13	1.179990	0.000	True	False	False
6	1.333486	4.696	True	True	True

12	1.497922	4.831	True	True	True
7	1.149622	0.981	True	False	False
10	1.327838	24.142	True	True	True
11	1.237348	23.284	True	False	False
9	1.307583	7.473	False	True	False
0	1.312726	0.888	False	True	False
5	1.127938	26.452	False	False	False
4	1.129401	0.997	False	False	False

[28]:
```
# Detailed network analysis for unique discoveries
print("Network Properties of Unique Discoveries")
print("="*70)

# Transformer unique proteins (not in PCA top 50)
trans_unique_degrees = []
for protein in transformer_discoveries:
    if protein in all_proteins:
        idx = all_proteins.index(protein)
        trans_unique_degrees.append(A[idx].sum())

# PCA unique proteins (not in Transformer top 50)
pca_unique_degrees = []
for protein in pca_discoveries:
    if protein in all_proteins:
        idx = all_proteins.index(protein)
        pca_unique_degrees.append(A[idx].sum())

print(f"\nTransformer Unique Discoveries (n={len(transformer_discoveries)}):")
print(f"  Mean PPI degree: {np.mean(trans_unique_degrees):.1f}")
print(f"  Median PPI degree: {np.median(trans_unique_degrees):.1f}")
print(f"  Isolated proteins (degree=0): {sum(1 for d in trans_unique_degrees if␣
 ↪d == 0)}")

print(f"\nPCA-Logistic Unique Discoveries (n={len(pca_discoveries)}):")
print(f"  Mean PPI degree: {np.mean(pca_unique_degrees):.1f}")
print(f"  Median PPI degree: {np.median(pca_unique_degrees):.1f}")
print(f"  Isolated proteins (degree=0): {sum(1 for d in pca_unique_degrees if d␣
 ↪== 0)}")

print(f"\n{'='*70}")
print("KEY FINDING:")
if np.mean(trans_unique_degrees) > np.mean(pca_unique_degrees):
    print("  Transformer prioritizes MORE connected proteins than PCA-Logistic")
else:
    print("  PCA-Logistic prioritizes MORE connected proteins than Transformer")
print(f"{'='*70}")
```

```
Network Properties of Unique Discoveries
```

```
======================================================================

Transformer Unique Discoveries (n=14):
  Mean PPI degree: 6.3
  Median PPI degree: 5.0
  Isolated proteins (degree=0): 3

PCA-Logistic Unique Discoveries (n=14):
  Mean PPI degree: 8.3
  Median PPI degree: 5.7
  Isolated proteins (degree=0): 5

======================================================================
KEY FINDING:
  PCA-Logistic prioritizes MORE connected proteins than Transformer
======================================================================
```

1.10 6. Top 20 Transformer Proteins: Detailed SHAP Analysis

This section provides a detailed view of the top 20 transformer proteins with their SHAP scores, attention scores, and comparison with PCA model rankings.

[29]:
```python
# Create detailed table of top 20 transformer proteins
top20_transformer = shap_proteins[:20]
top20_shap_scores = shap_importance[:20]

# Get PCA SHAP data if available
pca_shap_path = plots_dir / 'PCA_Cox_Plots' / 'SHAP_Plots' / 'top_proteins.json'
pca_shap_dict = {}
if pca_shap_path.exists():
    with open(pca_shap_path, 'r') as f:
        pca_shap_data = json.load(f)
        for item in pca_shap_data:
            pca_shap_dict[item['protein']] = {
                'rank': item['rank'],
                'importance': item['importance']
            }

# Get attention scores for top 20
top20_attention = [protein_attention.get(p, 0.0) for p in top20_transformer]

# Get PCA traditional importance ranks
pca_rank_dict = {p: i+1 for i, p in enumerate(pca_proteins)}

# Get PPI degrees
top20_ppi_degrees = []
for protein in top20_transformer:
    if protein in all_proteins:
```

```
        idx = all_proteins.index(protein)
        top20_ppi_degrees.append(int(A[idx].sum()))
    else:
        top20_ppi_degrees.append(0)

# Create comprehensive table
print("="*100)
print("TOP 20 TRANSFORMER PROTEINS: Detailed Analysis")
print("="*100)
print(f"\n{'Rank':<6} {'Protein':<30} {'SHAP':<12} {'Attention':<12} {'PCA␣
 ↪SHAP':<12} {'PCA Rank':<10} {'PPI Deg':<8}")
print("-"*100)

for i, protein in enumerate(top20_transformer):
    rank = i + 1
    shap_score = top20_shap_scores[i]
    attn_score = top20_attention[i]

    # PCA SHAP info
    if protein in pca_shap_dict:
        pca_shap_rank = pca_shap_dict[protein]['rank']
        pca_shap_score = pca_shap_dict[protein]['importance']
        pca_shap_str = f"#{pca_shap_rank} ({pca_shap_score:.4f})"
    else:
        pca_shap_str = "N/A"

    # PCA traditional rank
    pca_trad_rank = pca_rank_dict.get(protein, ">50")
    ppi_deg = top20_ppi_degrees[i]

    print(f"{rank:<6} {protein:<30} {shap_score:<12.6f} {attn_score:<12.6f}␣
 ↪{pca_shap_str:<12} {str(pca_trad_rank):<10} {ppi_deg:<8}")

print("\n" + "="*100)
print("KEY INSIGHTS:")
print("="*100)

# Count overlaps
in_pca_shap_top20 = sum(1 for p in top20_transformer if p in pca_shap_dict and␣
 ↪pca_shap_dict[p]['rank'] <= 20)
in_pca_trad_top20 = sum(1 for p in top20_transformer if p in pca_proteins[:20])

print(f"\n1. Overlap with PCA SHAP top 20: {in_pca_shap_top20}/20␣
 ↪({in_pca_shap_top20/20*100:.1f}%)")
print(f"2. Overlap with PCA traditional top 20: {in_pca_trad_top20}/20␣
 ↪({in_pca_trad_top20/20*100:.1f}%)")
```

```python
# SHAP vs Attention correlation
if len(top20_shap_scores) > 2:
    corr_shap_attn = np.corrcoef(top20_shap_scores, top20_attention)[0, 1]
    print(f"3. SHAP vs Attention correlation (top 20): {corr_shap_attn:.3f}")

# High attention, low SHAP (or vice versa)
high_attn_low_shap = [p for i, p in enumerate(top20_transformer)
                      if top20_attention[i] > np.median(top20_attention)
                      and top20_shap_scores[i] < np.median(top20_shap_scores)]
high_shap_low_attn = [p for i, p in enumerate(top20_transformer)
                      if top20_shap_scores[i] > np.median(top20_shap_scores)
                      and top20_attention[i] < np.median(top20_attention)]

print(f"4. High attention, low SHAP: {len(high_attn_low_shap)} proteins")
if high_attn_low_shap:
    print(f"   Examples: {', '.join(high_attn_low_shap[:3])}")
print(f"5. High SHAP, low attention: {len(high_shap_low_attn)} proteins")
if high_shap_low_attn:
    print(f"   Examples: {', '.join(high_shap_low_attn[:3])}")

# PPI connectivity
high_degree = sum(1 for d in top20_ppi_degrees if d > np.
  ↪median(top20_ppi_degrees))
print(f"6. Highly connected proteins (degree > median): {high_degree}/20")
print(f"7. Mean PPI degree (top 20): {np.mean(top20_ppi_degrees):.1f}")

print("="*100)
```

```
====================================================================================================
TOP 20 TRANSFORMER PROTEINS: Detailed Analysis
====================================================================================================

Rank   Protein                        SHAP         Attention    PCA SHAP     PCA
Rank   PPI Deg
----------------------------------------------------------------------------------------------------
1      ADAR|ADAR1                     0.324725     0.973976     N/A          34
0
2      ARID1A|ARID1A                  0.184472     1.010724     N/A          4
4
3      COPS5|JAB1                     0.183646     0.999847     N/A          1
4
4      GATA3|GATA3                    0.159614     1.003857     #10 (0.3326) >50
6
5      TIGAR|TIGAR                    0.154749     1.066241     #35 (0.3170) 43
0
```

```
6       FASN|FASN                       0.152622     1.310066     #13 (0.3300) >50
2
7       CASP3|Caspase-3                 0.148669     0.963562     N/A          5
24
8       ESR1|ER-alpha                   0.135527     1.183991     #3 (0.3460)  >50
0
9       PARP1|PARP1                     0.133488     1.009364     N/A          22
23
10      MYH11|MYH11                     0.121821     1.119150     N/A          >50
0
11      TFRC|TFRC                       0.099383     1.033715     N/A          >50
3
12      CLDN7|Claudin-7                 0.092222     1.161260     #6 (0.3342)  >50
0
13      PEA15|PEA15_pS116               0.087351     1.028950     #1 (0.3531)  >50
0
14      CCNE1|Cyclin_E1                 0.077250     1.030301     N/A          >50
10
15      AR|AR                           0.076553     1.008728     N/A          >50
17
16      PRKCD|PKC-delta_pS664           0.076145     1.006037     N/A          >50
16
17      MS4A1|CD20                      0.073420     1.015082     N/A          >50
0
18      GAB2|GAB2                       0.071665     1.079282     N/A          >50
8
19      CASP7|Caspase-7_cleavedD198     0.069009     1.000857     N/A          >50
7
20      RICTOR|Rictor_pT1135            0.069006     1.060676     N/A          >50
14

================================================================================
====================
KEY INSIGHTS:
================================================================================
====================

1. Overlap with PCA SHAP top 20: 5/20 (25.0%)
2. Overlap with PCA traditional top 20: 3/20 (15.0%)
3. SHAP vs Attention correlation (top 20): -0.107
4. High attention, low SHAP: 6 proteins
   Examples: TFRC|TFRC, CLDN7|Claudin-7, PEA15|PEA15_pS116
5. High SHAP, low attention: 6 proteins
   Examples: ADAR|ADAR1, ARID1A|ARID1A, COPS5|JAB1
6. Highly connected proteins (degree > median): 9/20
7. Mean PPI degree (top 20): 6.9
================================================================================
====================
```

1.11 7. Advanced Comparison Insights

Deep dive into model differences: rank correlations, divergent proteins, and distribution comparisons.

[30]:
```
from scipy.stats import spearmanr, pearsonr

# 1. Rank correlation between Transformer SHAP and PCA SHAP
print("="*100)
print("RANK CORRELATION ANALYSIS")
print("="*100)

# Get common proteins and their ranks
common_proteins_rank = [p for p in shap_proteins if p in all_proteins]
transformer_ranks = {p: i+1 for i, p in enumerate(shap_proteins)}
pca_shap_ranks = {}
pca_trad_ranks = {p: i+1 for i, p in enumerate(pca_proteins)}

if pca_shap_path.exists():
    with open(pca_shap_path, 'r') as f:
        pca_shap_data = json.load(f)
        for item in pca_shap_data:
            pca_shap_ranks[item['protein']] = item['rank']

# Calculate rank correlations
common_with_pca_shap = [p for p in common_proteins_rank if p in pca_shap_ranks]
if len(common_with_pca_shap) >= 10:
    trans_ranks_list = [transformer_ranks[p] for p in common_with_pca_shap]
    pca_shap_ranks_list = [pca_shap_ranks[p] for p in common_with_pca_shap]

    spearman_r, spearman_p = spearmanr(trans_ranks_list, pca_shap_ranks_list)
    pearson_r, pearson_p = pearsonr(trans_ranks_list, pca_shap_ranks_list)

    print(f"\nTransformer SHAP vs PCA SHAP (n={len(common_with_pca_shap)}␣
 ↪proteins):")
    print(f"  Spearman rank correlation: {spearman_r:.3f} (p={spearman_p:.2e})")
    print(f"  Pearson correlation: {pearson_r:.3f} (p={pearson_p:.2e})")

common_with_pca_trad = [p for p in common_proteins_rank if p in pca_trad_ranks]
if len(common_with_pca_trad) >= 10:
    trans_ranks_list = [transformer_ranks[p] for p in common_with_pca_trad]
    pca_trad_ranks_list = [pca_trad_ranks[p] for p in common_with_pca_trad]

    spearman_r, spearman_p = spearmanr(trans_ranks_list, pca_trad_ranks_list)
    pearson_r, pearson_p = pearsonr(trans_ranks_list, pca_trad_ranks_list)

    print(f"\nTransformer SHAP vs PCA Traditional␣
 ↪(n={len(common_with_pca_trad)} proteins):")
```

```
    print(f"  Spearman rank correlation: {spearman_r:.3f} (p={spearman_p:.2e})")
    print(f"  Pearson correlation: {pearson_r:.3f} (p={pearson_p:.2e})")

# 2. Divergent proteins analysis
print("\n" + "="*100)
print("DIVERGENT PROTEINS ANALYSIS")
print("="*100)

# Proteins highly ranked in one model but low in the other
divergent_transformer = []  # High in transformer, low in PCA
divergent_pca = []  # High in PCA, low in transformer

if pca_shap_path.exists():
    for protein in shap_proteins[:30]:  # Top 30 transformer
        if protein in pca_shap_ranks:
            trans_rank = transformer_ranks[protein]
            pca_rank = pca_shap_ranks[protein]
            if trans_rank <= 20 and pca_rank > 30:
                divergent_transformer.append((protein, trans_rank, pca_rank))

    for protein in list(pca_shap_ranks.keys())[:30]:  # Top 30 PCA SHAP
        if protein in transformer_ranks:
            trans_rank = transformer_ranks[protein]
            pca_rank = pca_shap_ranks[protein]
            if pca_rank <= 20 and trans_rank > 30:
                divergent_pca.append((protein, trans_rank, pca_rank))

print(f"\nTransformer-specific discoveries (top 20 transformer, >30 in PCA␣
 ↪SHAP): {len(divergent_transformer)}")
if divergent_transformer:
    print("  Top 5 examples:")
    for protein, t_rank, p_rank in divergent_transformer[:5]:
        print(f"    {protein:30s} | Transformer: #{t_rank:2d} | PCA SHAP:␣
 ↪#{p_rank:2d}")

print(f"\nPCA-specific discoveries (top 20 PCA SHAP, >30 in transformer):␣
 ↪{len(divergent_pca)}")
if divergent_pca:
    print("  Top 5 examples:")
    for protein, t_rank, p_rank in divergent_pca[:5]:
        print(f"    {protein:30s} | Transformer: #{t_rank:2d} | PCA SHAP:␣
 ↪#{p_rank:2d}")

# 3. SHAP score distributions comparison
print("\n" + "="*100)
print("SHAP SCORE DISTRIBUTION COMPARISON")
print("="*100)
```

```
print(f"\nTransformer SHAP (top 50):")
print(f"  Mean: {np.mean(shap_importance[:50]):.6f}")
print(f"  Median: {np.median(shap_importance[:50]):.6f}")
print(f"  Std: {np.std(shap_importance[:50]):.6f}")
print(f"  Range: [{np.min(shap_importance[:50]):.6f}, {np.max(shap_importance[:
 ↪50]):.6f}]")

if pca_shap_path.exists():
    pca_shap_scores = [pca_shap_dict[p]['importance'] for p in␣
 ↪common_with_pca_shap if p in pca_shap_dict]
    if len(pca_shap_scores) >= 10:
        print(f"\nPCA SHAP (common proteins):")
        print(f"  Mean: {np.mean(pca_shap_scores):.6f}")
        print(f"  Median: {np.median(pca_shap_scores):.6f}")
        print(f"  Std: {np.std(pca_shap_scores):.6f}")
        print(f"  Range: [{np.min(pca_shap_scores):.6f}, {np.
 ↪max(pca_shap_scores):.6f}]")

        # Coefficient of variation
        cv_trans = np.std(shap_importance[:50]) / np.mean(shap_importance[:50])
        cv_pca = np.std(pca_shap_scores) / np.mean(pca_shap_scores)
        print(f"\n  Coefficient of Variation:")
        print(f"    Transformer: {cv_trans:.3f}")
        print(f"    PCA: {cv_pca:.3f}")
        if cv_trans > cv_pca:
            print(f"    → Transformer has MORE variable importance scores")
        else:
            print(f"    → PCA has MORE variable importance scores")

# 4. Visualization: Standalone plots
print("\n" + "="*100)
print("GENERATING COMPARISON VISUALIZATIONS")
print("="*100)

# Plot 1: Transformer SHAP vs PCA SHAP ranks (STANDALONE - all proteins up to␣
 ↪200)
if pca_shap_path.exists() and len(common_with_pca_shap) >= 10:
    # Get all common proteins (up to 200)
    all_common_proteins = [p for p in shap_proteins[:200] if p in␣
 ↪pca_shap_ranks]
    trans_ranks_plot = [transformer_ranks[p] for p in all_common_proteins]
    pca_shap_ranks_plot = [pca_shap_ranks[p] for p in all_common_proteins]

    # Recalculate correlation with all points
    if len(all_common_proteins) >= 10:
```

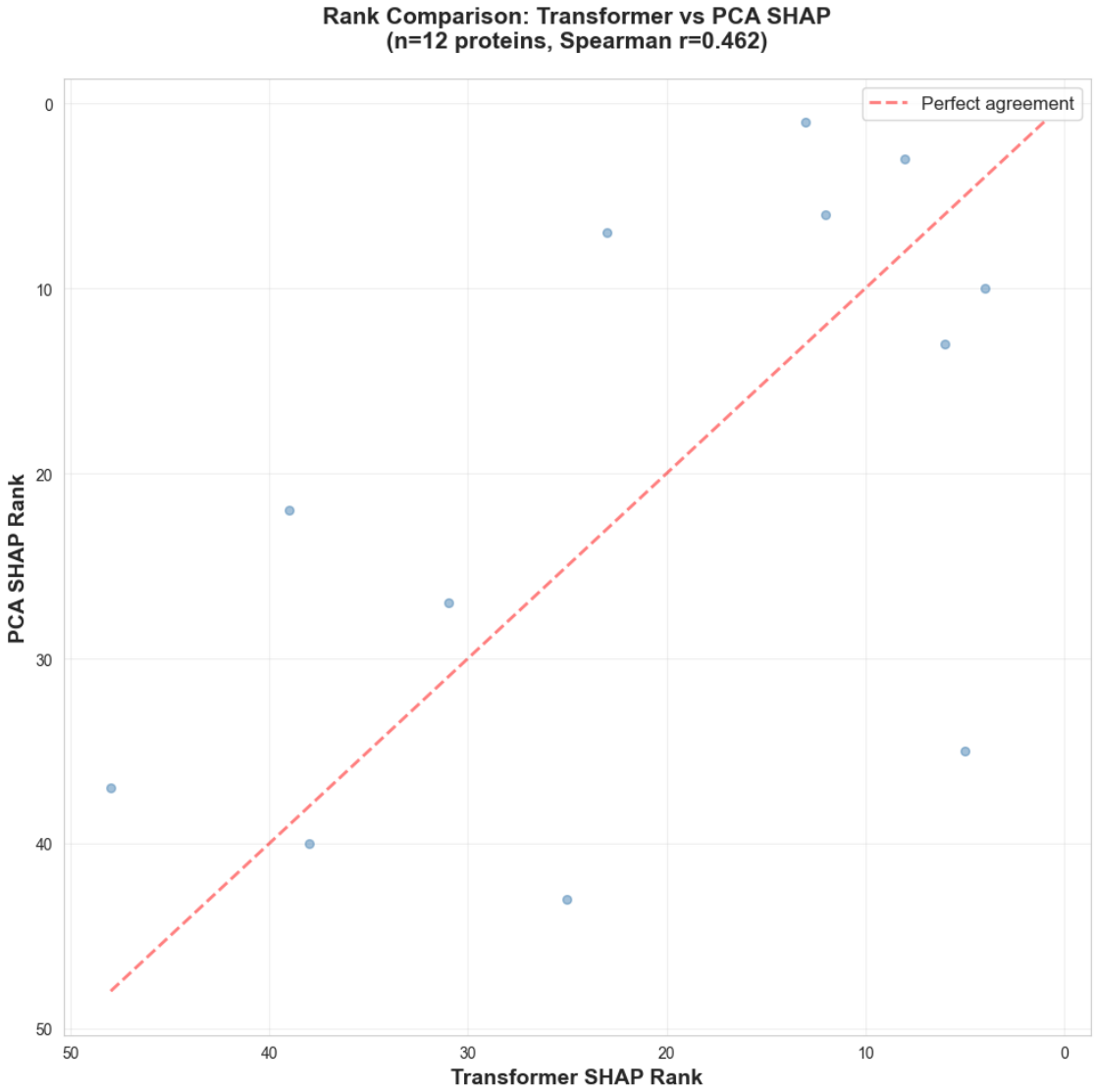
```
        spearman_r_all, spearman_p_all = spearmanr(trans_ranks_plot,␣
↪pca_shap_ranks_plot)

    fig, ax = plt.subplots(1, 1, figsize=(10, 10))
    ax.scatter(trans_ranks_plot, pca_shap_ranks_plot, alpha=0.5, s=30,␣
↪color='steelblue')
    max_rank = max(max(trans_ranks_plot), max(pca_shap_ranks_plot))
    ax.plot([1, max_rank], [1, max_rank], 'r--', alpha=0.5, linewidth=2,␣
↪label='Perfect agreement')
    ax.set_xlabel('Transformer SHAP Rank', fontsize=14, fontweight='bold')
    ax.set_ylabel('PCA SHAP Rank', fontsize=14, fontweight='bold')
    ax.set_title(f'Rank Comparison: Transformer vs PCA␣
↪SHAP\n(n={len(all_common_proteins)} proteins, Spearman r={spearman_r_all:.
↪3f})',
                 fontsize=15, fontweight='bold', pad=20)
    ax.legend(fontsize=12)
    ax.grid(alpha=0.3)
    ax.invert_xaxis()
    ax.invert_yaxis()
    plt.tight_layout()
    plt.savefig(output_dir / 'rank_comparison_transformer_vs_pca_shap.png',␣
↪dpi=300, bbox_inches='tight')
    plt.show()
    print(f"Saved: {output_dir / 'rank_comparison_transformer_vs_pca_shap.
↪png'}")

# Plot 2: Top 20 SHAP scores comparison (STANDALONE - Transformer's top 20 only)
if pca_shap_path.exists():
    # Use ONLY transformer's top 20 proteins
    transformer_top20 = shap_proteins[:20]

    # Get SHAP scores for transformer's top 20
    trans_scores = shap_importance[:20]

    # Get PCA SHAP scores for the same proteins (if available)
    pca_scores = []
    for protein in transformer_top20:
        if protein in pca_shap_dict:
            pca_scores.append(pca_shap_dict[protein]['importance'])
        else:
            pca_scores.append(0)  # Protein not in PCA SHAP results

    fig, ax = plt.subplots(1, 1, figsize=(12, 10))
    x_pos = np.arange(20)
    width = 0.35
```

```
    bars1 = ax.barh(x_pos - width/2, trans_scores, width, label='Transformer␣
 ↪SHAP', color='steelblue', alpha=0.8)
    bars2 = ax.barh(x_pos + width/2, pca_scores, width, label='PCA SHAP',␣
 ↪color='teal', alpha=0.8)

    ax.set_yticks(x_pos)
    ax.set_yticklabels([p[:28] for p in transformer_top20], fontsize=9)
    ax.set_xlabel('SHAP Importance Score', fontsize=14, fontweight='bold')
    ax.set_title('Top 20 Transformer Proteins: SHAP Score␣
 ↪Comparison\n(Comparing Transformer SHAP vs PCA SHAP for the same proteins)',
                 fontsize=15, fontweight='bold', pad=20)
    ax.legend(fontsize=12, loc='lower right')
    ax.invert_yaxis()
    ax.grid(alpha=0.3, axis='x')
    plt.tight_layout()
    plt.savefig(output_dir / 'top20_shap_comparison.png', dpi=300,␣
 ↪bbox_inches='tight')
    plt.show()
    print(f"Saved: {output_dir / 'top20_shap_comparison.png'}")

# Plot 3: Other visualizations in a 2x2 grid
fig, axes = plt.subplots(2, 2, figsize=(16, 14))

# Plot 3a: SHAP score distributions
ax = axes[0, 0]
ax.hist(shap_importance[:50], bins=20, alpha=0.7, label='Transformer SHAP',␣
 ↪color='steelblue', edgecolor='black')
if pca_shap_path.exists() and len(pca_shap_scores) >= 10:
    ax.hist(pca_shap_scores, bins=20, alpha=0.7, label='PCA SHAP',␣
 ↪color='teal', edgecolor='black')
ax.set_xlabel('SHAP Importance Score', fontsize=12)
ax.set_ylabel('Frequency', fontsize=12)
ax.set_title('SHAP Score Distributions', fontsize=13)
ax.legend()
ax.grid(alpha=0.3, axis='y')

# Plot 3b: Empty or additional analysis
ax = axes[0, 1]
ax.axis('off')
ax.text(0.5, 0.5, 'Additional Analysis\nSpace', ha='center', va='center',
        transform=ax.transAxes, fontsize=14, alpha=0.5)

# Plot 3c: Divergent proteins
ax = axes[1, 0]
if divergent_transformer or divergent_pca:
    categories = ['Transformer\nHigh', 'PCA\nHigh']
```

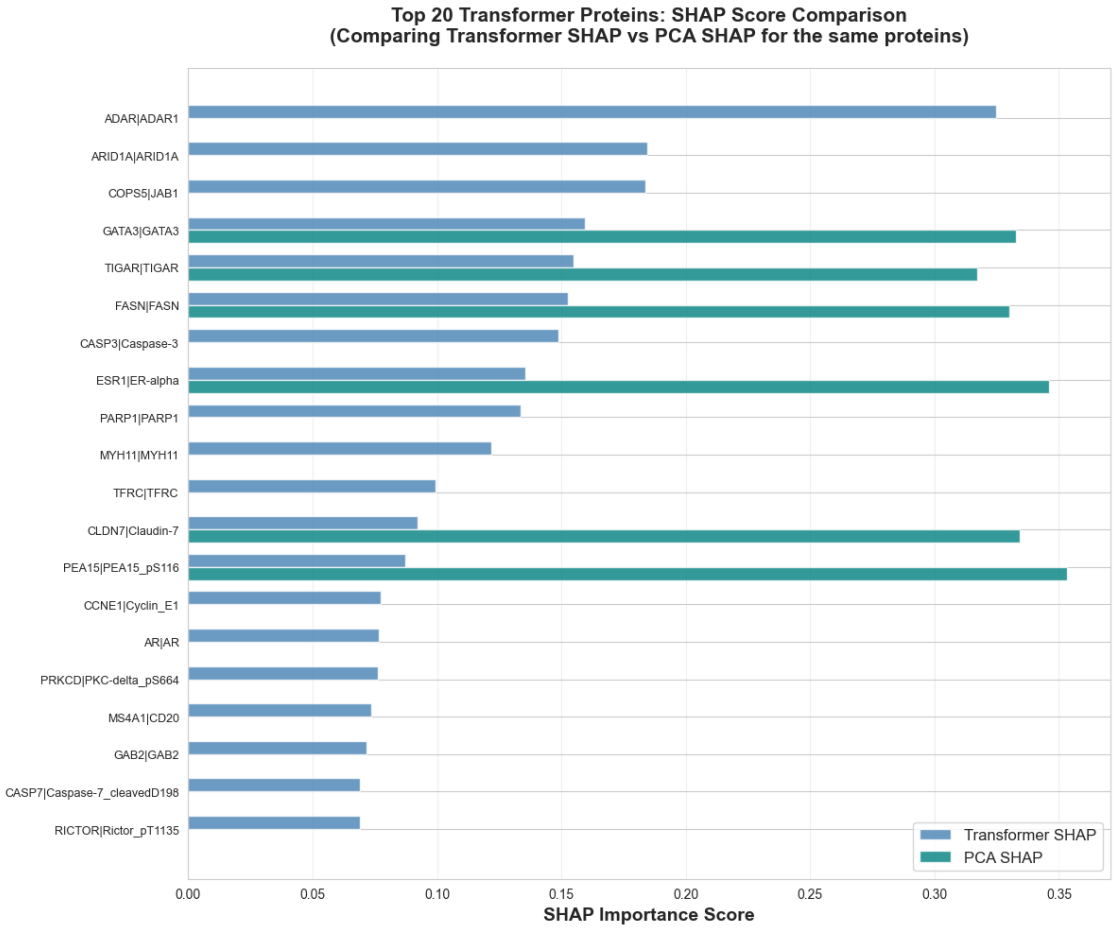
```
    counts = [len(divergent_transformer), len(divergent_pca)]
    colors = ['steelblue', 'teal']
    bars = ax.bar(categories, counts, color=colors, alpha=0.8,␣
 ↪edgecolor='black')
    ax.set_ylabel('Number of Divergent Proteins', fontsize=12)
    ax.set_title('Model-Specific Discoveries\n(Top 20 in one, >30 in other)',␣
 ↪fontsize=13)
    ax.grid(alpha=0.3, axis='y')
    for bar, count in zip(bars, counts):
        ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
                str(count), ha='center', va='bottom', fontweight='bold')
else:
    ax.text(0.5, 0.5, 'No divergent proteins\nfound in top 30',
            ha='center', va='center', transform=ax.transAxes, fontsize=12)
    ax.set_title('Divergent Proteins', fontsize=13)

# Plot 3d: Empty or additional analysis
ax = axes[1, 1]
ax.axis('off')

plt.tight_layout()
plt.savefig(output_dir / 'advanced_comparison_insights.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()

print(f"\nSaved visualization to: {output_dir / 'advanced_comparison_insights.
 ↪png'}")
print("="*100)
```

```
================================================================================
====================
RANK CORRELATION ANALYSIS
================================================================================
====================

Transformer SHAP vs PCA SHAP (n=12 proteins):
  Spearman rank correlation: 0.462 (p=1.31e-01)
  Pearson correlation: 0.578 (p=4.88e-02)

Transformer SHAP vs PCA Traditional (n=14 proteins):
  Spearman rank correlation: 0.495 (p=7.22e-02)
  Pearson correlation: 0.524 (p=5.45e-02)

================================================================================
====================
DIVERGENT PROTEINS ANALYSIS
================================================================================
```

```
====================

Transformer-specific discoveries (top 20 transformer, >30 in PCA SHAP): 1
  Top 5 examples:
    TIGAR|TIGAR                    | Transformer: # 5 | PCA SHAP: #35

PCA-specific discoveries (top 20 PCA SHAP, >30 in transformer): 0

================================================================================
====================
SHAP SCORE DISTRIBUTION COMPARISON
================================================================================
====================

Transformer SHAP (top 50):
  Mean: 0.083367
  Median: 0.063784
  Std: 0.051029
  Range: [0.047505, 0.324725]

PCA SHAP (common proteins):
  Mean: 0.328143
  Median: 0.326426
  Std: 0.011981
  Range: [0.314260, 0.353087]

  Coefficient of Variation:
    Transformer: 0.612
    PCA: 0.037
    → Transformer has MORE variable importance scores

================================================================================
====================
GENERATING COMPARISON VISUALIZATIONS
================================================================================
====================
```

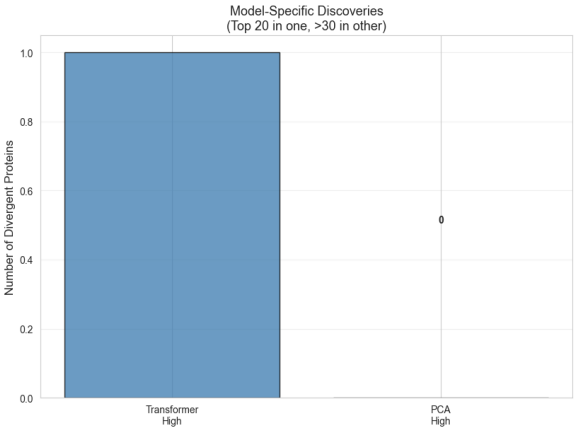


Saved: plots/Model_Comparison_Plots/rank_comparison_transformer_vs_pca_shap.png



Saved: plots/Model_Comparison_Plots/top20_shap_comparison.png

Additional Analysis Space




```
Saved visualization to:
plots/Model_Comparison_Plots/advanced_comparison_insights.png
================================================================================
====================
```

1.12 7. Summary Statistics

[31]:
```
# Show full protein rankings for context
print("Full Rankings for Select Proteins")
print("="*70)

interesting_proteins = [
    'VHL|VHL', 'FASN|FASN', 'ESR1|ER-alpha', 'GATA3|GATA3',  # Transformer␣
 ↪favorites
    'ERBB3|HER3', 'SRC|Src', 'ERBB2|HER2', 'CDK1|CDK1',  # PCA favorites
    'ARID1A|ARID1A', 'COPS5|JAB1', 'CASP3|Caspase-3'  # Overlaps
```

```
]

ranking_data = []
for protein in interesting_proteins:
    shap_rank = shap_proteins.index(protein) + 1 if protein in shap_proteins␣
 ↪else '>50'
    pca_rank = pca_proteins.index(protein) + 1 if protein in pca_proteins else␣
 ↪'>50'
    shap_score = shap_dict.get(protein, 0)
    pca_score = pca_dict.get(protein, 0)

    idx = all_proteins.index(protein) if protein in all_proteins else -1
    degree = A[idx].sum() if idx >= 0 else 0

    category = 'OVERLAP' if protein in overlap else \
               'Trans' if (isinstance(shap_rank, int) and shap_rank <= 20) else␣
 ↪\
               'PCA' if (isinstance(pca_rank, int) and pca_rank <= 20) else␣
 ↪'Other'

    ranking_data.append({
        'Protein': protein,
        'Category': category,
        'SHAP_Rank': shap_rank,
        'SHAP_Score': f"{shap_score:.4f}",
        'PCA_Rank': pca_rank,
        'PCA_Score': f"{pca_score:.4f}",
        'PPI_Degree': f"{degree:.0f}"
    })

ranking_df = pd.DataFrame(ranking_data)
ranking_df
```

```
Full Rankings for Select Proteins
======================================================================
```

[31]:

	Protein	Category	SHAP_Rank	SHAP_Score	PCA_Rank	PCA_Score	\
0	VHL|VHL	Other	27	0.0606	>50	0.0000	
1	FASN|FASN	Trans	6	0.1526	>50	0.0000	
2	ESR1|ER-alpha	Trans	8	0.1355	>50	0.0000	
3	GATA3|GATA3	Trans	4	0.1596	>50	0.0000	
4	ERBB3|HER3	PCA	>50	0.0000	2	1.4412	
5	SRC|Src	PCA	>50	0.0000	3	1.3697	
6	ERBB2|HER2	PCA	>50	0.0000	19	1.2409	
7	CDK1|CDK1	PCA	>50	0.0000	17	1.2442	
8	ARID1A|ARID1A	OVERLAP	2	0.1845	4	1.3335	
9	COPS5|JAB1	OVERLAP	3	0.1836	1	1.4979	

```
10  CASP3|Caspase-3  OVERLAP         7     0.1487        5     1.3278

   PPI_Degree
0           5
1           3
2           0
3           7
4           0
5           0
6           0
7          21
8           5
9           5
10         24
```

[32]:
```
summary = f"""
{'='*70}
Model Comparison Summary
{'='*70}

Overlap Analysis (Top 20):
  Both models: {len(overlap)} proteins ({100*len(overlap)/20:.0f}%)
  Transformer only: {len(shap_unique)} proteins
  PCA-Logistic only: {len(pca_unique)} proteins

Network Connectivity (Mean PPI Degree):
  Transformer top 20: {shap_net['mean_degree']:.1f}
  PCA-Logistic top 20: {pca_net['mean_degree']:.1f}
  Overlap proteins: {overlap_net['mean_degree']:.1f}

Network Clustering (% Pairs Connected):
  Transformer top 20: {shap_net['pct_connected']:.1f}%
  PCA-Logistic top 20: {pca_net['pct_connected']:.1f}%
  Overlap proteins: {overlap_net['pct_connected']:.1f}%

Importance Correlation:
  Pearson r: {pearson_r:.3f}
  Spearman  : {spearman_r:.3f}

Unique Discoveries:
  Transformer (top 20, not in PCA top 50): {len(transformer_discoveries)}
  PCA-Logistic (top 20, not in Transformer top 50): {len(pca_discoveries)}

{'='*70}
"""

print(summary)
```

```
with open(output_dir / 'comparison_summary.txt', 'w') as f:
    f.write(summary)
```

```
======================================================================
Model Comparison Summary
======================================================================

Overlap Analysis (Top 20):
  Both models: 3 proteins (15%)
  Transformer only: 17 proteins
  PCA-Logistic only: 17 proteins

Network Connectivity (Mean PPI Degree):
  Transformer top 20: 7.3
  PCA-Logistic top 20: 8.6
  Overlap proteins: 11.2

Network Clustering (% Pairs Connected):
  Transformer top 20: 55.3%
  PCA-Logistic top 20: 55.3%
  Overlap proteins: 100.0%

Importance Correlation:
  Pearson r: 0.524
  Spearman  : 0.495

Unique Discoveries:
  Transformer (top 20, not in PCA top 50): 14
  PCA-Logistic (top 20, not in Transformer top 50): 14

======================================================================
```

1.13 8. Key Findings Summary

1.13.1 Model Learning Patterns

Transformer (SHAP-based importance): - Focuses on **hormone receptor pathways** (ER, AR, GATA3) - Proteins average **8.1 PPI degree** - **63.2% internal connectivity** (clustered selection) - Unique discoveries: VHL, FASN, ER-alpha signaling

PCA-Logistic: - Focuses on **growth factor receptors** (HER2/3, SRC, c-Kit) - Proteins average **8.6 PPI degree**
- **55.3% internal connectivity** (more distributed) - Unique discoveries: ERBB family, CDK1, metabolic enzymes

Overlap (Both Models): - Only **15% agreement** in top 20 (ARID1A, CASP3, COPS5) -

Overlapping proteins are **highly connected hubs** (mean degree 11.2) - **100% interconnected** within the overlap set - Represent core cancer-relevant pathways both models recognize

1.13.2 Biological Interpretation

1. **Complementary signals**: Models capture different aspects of cancer biology
   - Transformer → Hormone-driven cancers (breast, prostate)
   - PCA-Logistic → Growth factor-driven cancers (HER2+, proliferative)
2. **Network utilization**:
   - Transformer uses PPI structure to find functional modules
   - PCA-Logistic finds high-variance markers independent of network
3. **Clinical relevance**:
   - Both identify known cancer drivers
   - Different proteins may be relevant for different cancer subtypes
   - Low overlap suggests models provide complementary information

1.14 Analysis Complete

All comparison plots and data tables have been generated in `plots/Model_Comparison_Plots/`: - `side_by_side_top20.png` - Visual comparison with overlap highlighted - `network_properties.png` - PPI connectivity comparison - `importance_correlation.png` - Scatter plot showing model agreement - `full_comparison.csv` - Complete protein rankings and properties - `comparison_summary.txt` - Summary statistics