performance and exaplainability

In [1]:
```
# install dependancies
import glob
import json
import numpy as np
import os
import pandas as pd
import re
import seaborn as sns
import matplotlib.pyplot as plt
from IPython.display import display
```

In [2]:
```
# Load in latest model hyperparameter data
param_files = glob.glob(os.path.join('../data/model_artifacts/', 'xgb_best_params_*.json'))
pattern = r'xgb_best_params_(\d{8})_(\d{6})\.json'
latest_file = max(
    param_files,
    key=lambda f: ''.join(re.findall(pattern, os.path.basename(f))[0])
    if re.findall(pattern, os.path.basename(f)) else ''
)

# Load the parameters
with open(latest_file, 'r') as f:
    params_data = json.load(f)

# Convert to DataFrame

model_params = (pd.DataFrame([params_data])
                if isinstance(params_data, dict)
                else pd.DataFrame(params_data)
)

# read in model metrics data
overall_metrics = pd.read_parquet('../data/03_overall_metrics.parquet')
per_class_metrics = pd.read_parquet('../data/03_per_class_metrics.parquet')
confusion_matrix  = pd.read_parquet('../data/03_confusion_matrix.parquet')

# read in explainablity data
global_importance = pd.read_parquet('../data/03_global_importance.parquet')
```

model hyperparameters

In [3]:
```
print("best model parameterd identified via RandomSearchCV")
display(model_params)
```

```
best model parameterd identified via RandomSearchCV
```

	subsample	n_estimators	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
0	0.8	300	1	8	0.05	0	0.9

The hyperparameters were obtained using Scikit-learn's RandomizedSearchCV rather than a full GridSearchCV due to computational constraints with the high-dimensional dataset. The search used 5-fold cross-validation on a stratified 10,000-sample subset of the data to ensure representation across all 28 product categories. The randomized approach explored 25 iterations from the parameter space, executing in parallel across 8 threads (half of the available i9-9900k 16-thread processor capacity with n_jobs=num_threads//2). Despite limiting thread allocation, the internal parallelization of XGBoost models pushed CPU utilization above 95% throughout the process, with processor temperatures exceeding 95°C. This approach provided an efficient balance between exploration and exploitation, completing in approximately 16 hours overnight versus the estimated 4-5 days a full grid search would have required.

The optimized hyperparameters collectively reflect a model tuned for this specific product classification challenge. With n_estimators=300 and max_depth=8, the model favors complexity necessary for distinguishing between 28 product categories using heterogeneous inputs. The moderate learning_rate=0.05 combined with min_child_weight=1 and gamma=0 suggests the feature engineering effectively reduced noise, allowing the model to confidently utilize subtle patterns. Meanwhile, subsample=0.8 and colsample_bytree=0.9 provide just enough regularization to prevent overfitting on the high-dimensional text embeddings while preserving signal from the carefully engineered features. This balanced configuration achieved 90-91% accuracy on the hold-out test set without overfitting, demonstrating robust generalization across product categories.

n_estimators

- Range tested: [100, 200, 300]
- Best value: 300
- Rationale:
    - Needed for 28-class problem with high-dimensional features (BERT embeddings, encoded details)
    - Captures subtle distinctions between similar product categories
    - Performance gain outweighed the increased training time

max_depth

- Range tested: [4, 6, 8]
- Best value: 8
- Rationale:
    - Captures complex relationships between text embeddings and product details
    - Necessary for modeling hierarchical product category structure
    - Handles signal complexity maintained after dimensionality reduction

learning_rate

- Range tested: [0.01, 0.05, 0.1]
- Best value: 0.05
- Rationale:
    - Balances convergence speed with stability for heterogeneous feature space
    - Prevents overweighting noisy signals from sparse product attributes
    - Optimal for integrating text embeddings with binary feature signals

min_child_weight

- Range tested: [1, 3, 5]
- Best value: 1
- Rationale:
    - Allows model to capture rare but distinctive product features

- Important for minority product categories with unique characteristics
- Benefits from detailed feature engineering that removed most noise

gamma

- Range tested: [0, 0.1, 0.2]
- Best value: 0
- Rationale:
  - Captures subtle distinctions between related product categories
  - No minimum loss reduction threshold needed due to effective preprocessing
  - Allows learning from sparse but relevant signals in features/detail fields

subsample

- Range tested: [0.8, 0.9]
- Best value: 0.8
- Rationale:
  - Reduces overfitting on highly specific product descriptions
  - Introduces randomness to improve generalization across product types
  - Handles imbalanced representation of different product categories

colsample_bytree

- Range tested: [0.8, 0.9]
- Best value: 0.9
- Rationale:
  - Uses most features while maintaining some diversity between trees
  - High value indicates most engineered features contribute meaningfully
  - Slightly reduced from 1.0 to prevent overfitting on text embedding dimensions

overall metrics

In [4]:

```
# output overall metrics
styled_overall = (overall_metrics
    .style
    .background_gradient(subset=['value'], cmap='YlGnBu', vmin=0, vmax=1)
    .bar(subset=['value'], color='#4a90e2', width=100, align='mid')
    .format({'value': '{:.4f}'})
    .set_caption('Model Performance Metrics (0-1 scale)')
)

display(styled_overall)
```

Model Performance Metrics (0-1 scale)

	metric	value
0	accuracy	0.9029
1	precision_macro	0.8771
2	recall_macro	0.7966
3	f1_macro	0.8207
4	precision_weighted	0.9055
5	recall_weighted	0.9029
6	f1_weighted	0.9008
7	top3_accuracy	0.9736
8	top5_accuracy	0.9875

- Strong overall accuracy (~90%)
- Excellent top-k accuracy metrics (top-3 >97%, top-5 >98%)
- Notable gap between macro and weighted metrics
- Macro recall is the lowest metric (~80%)
- Performance suggests common categories classified more accurately than rare ones
- F1 metrics confirm class imbalance impact on model performance

In [5]:

```
# output per_class metrics
styled_class_metrics = (per_class_metrics.drop('class_id', axis=1)
    .style
    .background_gradient(subset=['precision', 'recall', 'f1_score', 'accuracy'],
                        cmap='YlGnBu', vmin=0, vmax=1)
    .background_gradient(subset=['support'],
                        cmap='Oranges', vmin=0, vmax=per_class_metrics['support'].max())
    .bar(subset=['precision', 'recall', 'f1_score', 'accuracy'],
         color='#4a90e2', width=100, align='mid')
    .bar(subset=['support'],
         color='#ff9966', width=100, align='mid')
    .format({'precision': '{:.4f}', 'recall': '{:.4f}',
            'f1_score': '{:.4f}', 'accuracy': '{:.4f}'})
    .set_caption('Per-Class Performance Metrics')
)

display(styled_class_metrics)
```

Per-Class Performance Metrics

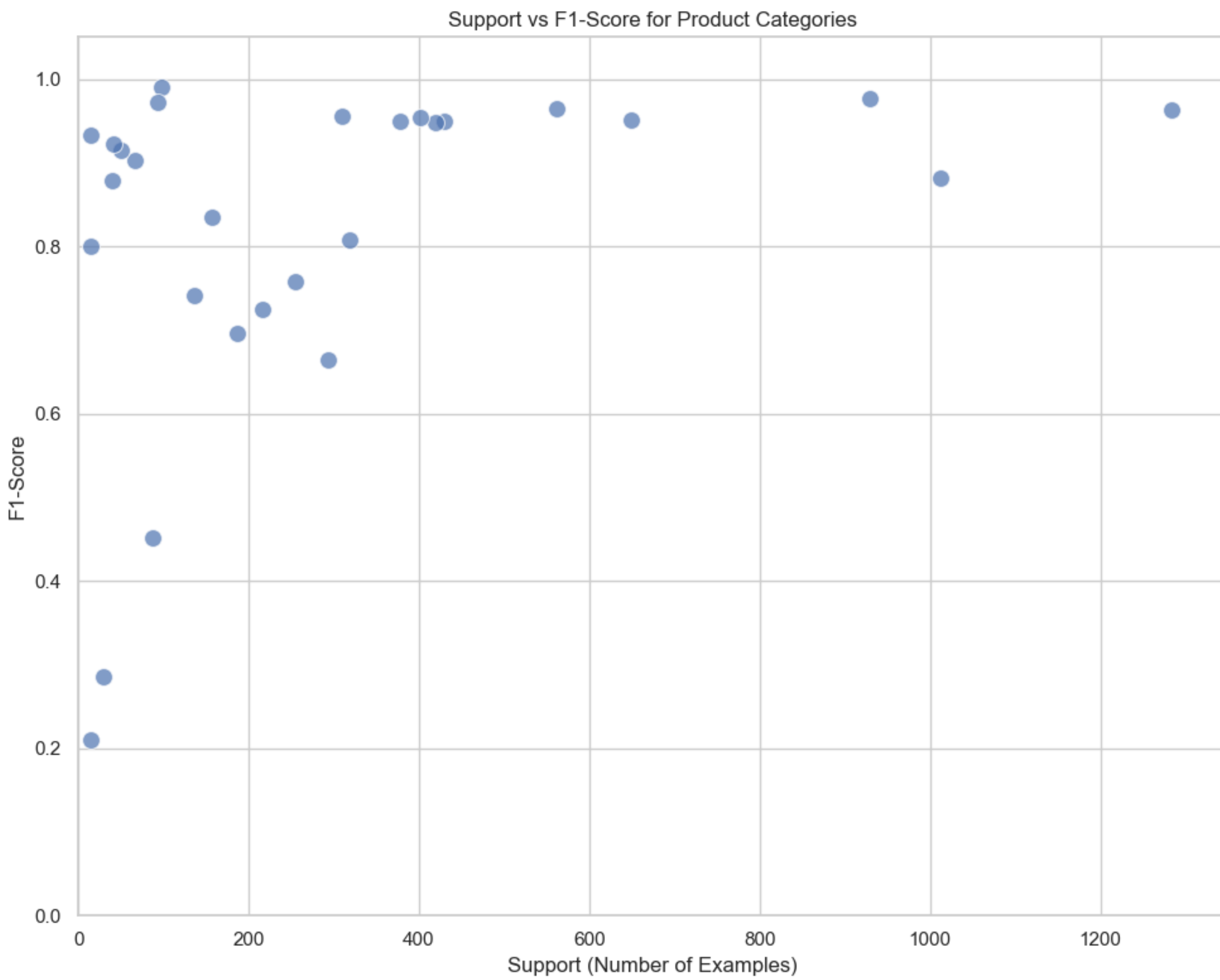
	class_name	precision	recall	f1_score	support	accuracy
11	Fashion	0.9520	0.9735	0.9626	1284	0.9735
15	Home	0.8387	0.9298	0.8819	1012	0.9298
4	Automotive	0.9733	0.9806	0.9770	930	0.9806
25	Tools & Home Improvement	0.9521	0.9492	0.9506	649	0.9492
24	Sports & Outdoors	0.9499	0.9786	0.9641	562	0.9786
0	All Beauty	0.9427	0.9558	0.9492	430	0.9558
19	Office Products	0.9519	0.9429	0.9474	420	0.9429
26	Toys & Games	0.9841	0.9254	0.9538	402	0.9254
8	Cell Phones & Accessories	0.9404	0.9603	0.9503	378	0.9603
17	Industrial & Scientific	0.8121	0.8019	0.8070	318	0.8019
13	Grocery	0.9700	0.9417	0.9557	309	0.9417
1	All Electronics	0.5663	0.8020	0.6638	293	0.8020
9	Computers	0.8564	0.6784	0.7571	255	0.6784
3	Arts, Crafts & Sewing	0.7880	0.6713	0.7250	216	0.6713
14	Health & Personal Care	0.7469	0.6505	0.6954	186	0.6505
20	Pet Supplies	0.9297	0.7580	0.8351	157	0.7580
6	Camera & Photo	0.8087	0.6838	0.7410	136	0.6838
10	Digital Music	0.9800	1.0000	0.9899	98	1.0000
18	Musical Instruments	0.9889	0.9570	0.9727	93	0.9570
16	Home Audio & Theater	0.6522	0.3448	0.4511	87	0.3448
5	Baby	0.9655	0.8485	0.9032	66	0.8485
2	Appliances	0.9773	0.8600	0.9149	50	0.8600
27	Video Games	1.0000	0.8571	0.9231	42	0.8571
22	Premium Beauty	0.8571	0.9000	0.8780	40	0.9000

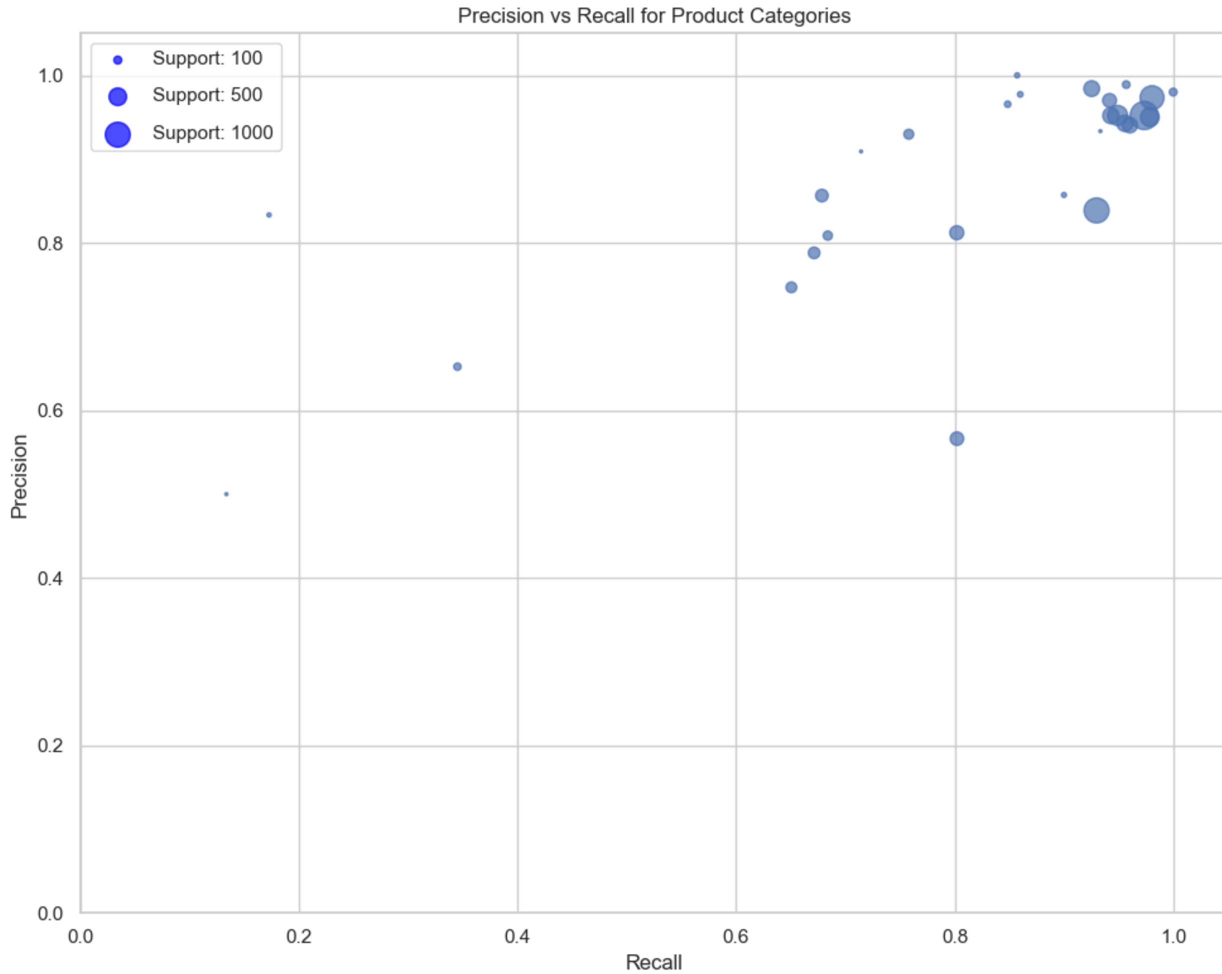
	class_name	precision	recall	f1_score	support	accuracy
7	Car Electronics	0.8333	0.1724	0.2857	29	0.1724
12	GPS & Navigation	0.5000	0.1333	0.2105	15	0.1333
21	Portable Audio & Accessories	0.9333	0.9333	0.9333	15	0.9333
23	Software	0.9091	0.7143	0.8000	14	0.7143

- Significant performance variation across categories, with some showing excellent metrics (>95%) while others struggle (<70%)
- Categories with larger support (sample size) generally perform better (Fashion, Automotive, Tools & Home Improvement)
- Several electronics-related categories underperform (All Electronics, Car Electronics, GPS & Navigation)
- Perfect recall (1.0) in niche categories like Digital Music and Portable Audio & Accessories
- Computers category shows low recall despite moderate support, suggesting frequent misclassification
- Software has perfect precision but poor recall, indicating high confidence when predicted but missed cases
- Home Audio & Theater shows notably weak performance across all metrics
- Industrial & Scientific category has balanced but mediocre metrics (~78%)

In [6]:
```
# plot support vs f1
sns.set_theme(style="whitegrid")

# Create scatter plot
plt.figure(figsize=(10, 8))
sns.scatterplot(
    data=per_class_metrics,
    x='support',
    y='f1_score',
    s=100,
    alpha=0.7
)

plt.title('Support vs F1-Score for Product Categories')
plt.xlabel('Support (Number of Examples)')
plt.ylabel('F1-Score')
plt.xlim(0, None)
plt.ylim(0, 1.05)
plt.tight_layout()
plt.show()

# plot prescision vs recall
plt.figure(figsize=(10, 8))
scatter = plt.scatter(
    x=per_class_metrics['recall'],
    y=per_class_metrics['precision'],
    s=per_class_metrics['support']/5,
    alpha=0.7
)
```

```
plt.title('Precision vs Recall for Product Categories')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.xlim(0, 1.05)
plt.ylim(0, 1.05)

# Add legend for bubble sizes
sizes = [100, 500, 1000]
for size in sizes:
    plt.scatter([], [], s=size/5, alpha=0.7, color='blue', label=f'Support: {size}')
plt.legend(scatterpoints=1, frameon=True, labelspacing=1)

plt.tight_layout()
plt.show()
```





support vs f1 relationship:

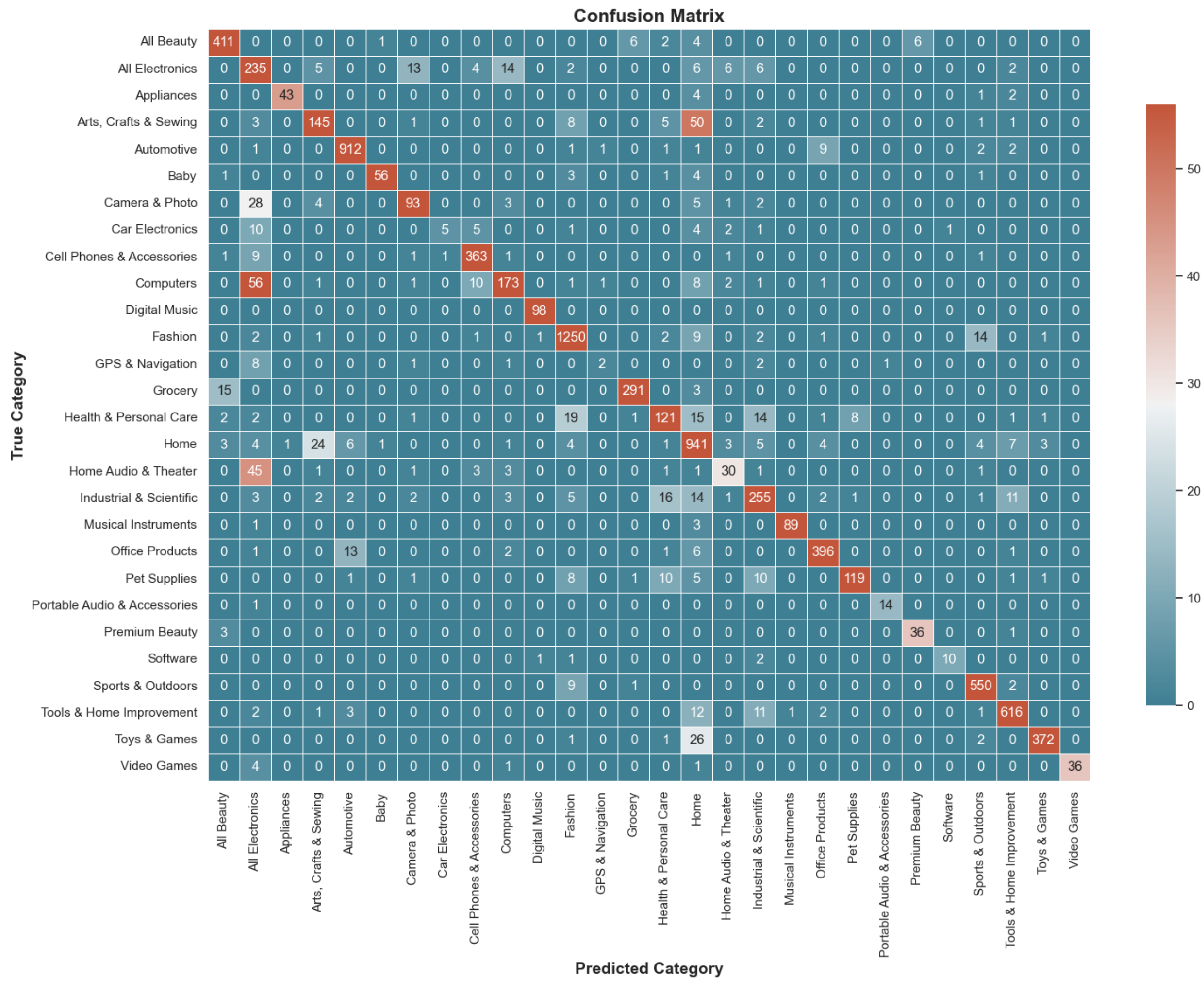
- There's a minimum viability threshold of examples needed (~200-300) before performance stabilizes
- The relationship suggests diminishing returns - adding more data to already well-performing categories likely won't yield significant improvements
- Some categories struggle despite having sufficient data, suggesting inherent classification difficulty rather than data scarcity
- The few extremely poorly performing categories (F1<0.3) likely have fundamental issues with feature representation or categorical confusion
- Categories with high performance despite low support indicate distinctive features making them easier to classify
- Data augmentation efforts should prioritize the lowest-performing categories with <300 examples
- The inconsistent mid-range performance suggests some categories might benefit from specialized feature engineering

precision vs recall vs support relationship:

- Model trade-off tendencies favor precision over recall in most categories
- Several high-support categories achieve both high precision and recall, indicating robust feature representation
- The few categories with low precision/high recall may have overly broad decision boundaries, causing false positives
- Categories with high precision/low recall likely have overly strict decision boundaries, missing valid

examples
- Larger support classes (bigger circles) generally cluster in the optimal upper-right quadrant, suggesting threshold tuning may be beneficial for smaller classes
- The model would benefit from threshold optimization for categories with imbalanced precision-recall metrics
- The scattered performance across categories with similar support sizes indicates inherent differences in category separability

In [7]:
```
# Get max value excluding diagonal elements
mask = np.eye(confusion_matrix.shape[0], dtype=bool)
max_non_diag = confusion_matrix.mask(mask).max().max()

plt.figure(figsize=(16, 12))

# Create a custom colormap similar to the correlation matrix
cmap = sns.diverging_palette(220, 20, as_cmap=True)

# Plot the heatmap with labeled axes
ax = sns.heatmap(confusion_matrix, annot=True, cmap=cmap, fmt=".0f",
            linewidths=.5, cbar_kws={"shrink": .8},
            vmax=max_non_diag)

# Add clear axis labels
plt.ylabel('True Category', fontsize=14, fontweight='bold')
plt.xlabel('Predicted Category', fontsize=14, fontweight='bold')
plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```



- Strong diagonal dominance indicates generally good classification performance
- Electronics-related categories show significant cross-confusion (All Electronics, Home Audio & Theater, Car Electronics)
- Arts/Crafts/Sewing and Health/Personal Care have substantial misclassifications
- Digital Music and Portable Audio show nearly perfect separation
- Home category experiences confusion with several related categories (Tools, Furniture, Office)
- Industrial & Scientific items get misclassified into diverse categories
- Computers often mispredicted as All Electronics (~51 misclassifications)
- Fashion category shows remarkably clear separation despite its large sample size
- Several categories with sufficient samples still experience confusion with related categories
- Pet Supplies shows multiple small confusions across various unrelated categories

explainabilty

global feature improtance

In [8]:
```
# print features with the highest global impact
styled_global_importance = (global_importance[global_importance['importance'] >= 0.1]
    .style
    .background_gradient(subset=['importance'], cmap='YlGnBu', vmin=0, vmax=1)
```

```
    .bar(subset=['importance'], color='#4a90e2', width=100, align='mid')
    .format({'importance': '{:.4f}'})
    .set_caption('SHAP value feature importance where >= 0.1')
)

display(styled_global_importance)
```

SHAP value feature importance where >= 0.1

	feature_name	importance
73	details_3	0.4443
52	details_10	0.2003
163	title_emb_2	0.1827
79	details_35	0.1783
97	details_7	0.1526
74	details_30	0.1515
66	details_23	0.1501
64	details_21	0.1283
174	title_emb_3	0.1201
51	details_1	0.1201
84	details_4	0.1134
96	details_6	0.1111
77	details_33	0.1035
57	details_15	0.1028

- Structured data (details fields) dominate the top features, with details_3 having by far the highest importance (0.46)
- Title embeddings contribute significantly with 3 components in the top 10 features
- The model heavily relies on a mix of product details and title semantics
- Feature embeddings appear less influential, with only one component in the top features
- The importance drops sharply after the top feature, then gradually decreases
- The reliance on multiple embedding components suggests the model captures semantic patterns rather than simple keywords
- The high importance of specific details fields indicates certain product attributes strongly correlate with categories

In [9]:
```
# sum contribution by original column type
rolled_df = global_importance.copy()

rolled_df['prefix'] = rolled_df['feature_name'].str.split('_').str[0]

# For features with prefix_emb pattern
mask = rolled_df['feature_name'].str.contains('_emb')
rolled_df.loc[mask, 'prefix'] = rolled_df.loc[mask, 'feature_name'].str.split('_emb').str[0] +

# Sum importances by prefix
rolled_df = rolled_df.groupby('prefix')['importance'].sum().reset_index()
```

```
rolled_df = rolled_df.rename(columns={'prefix': 'feature_group', 'importance': 'total_importan

# Sort by importance
rolled_df = rolled_df.sort_values('total_importance', ascending=False)

# style data frame
styled_rolled_df = (rolled_df
    .style
    .background_gradient(subset=['total_importance'], cmap='YlGnBu', vmin=0, vmax=1)
    .bar(subset=['total_importance'], color='#4a90e2', width=100, align='mid')
    .format({'total_importance': '{:.4f}'})
    .set_caption('SHAP value original feature importance')
)

display(styled_rolled_df)
```

SHAP value original feature importance

	feature_group	total_importance
1	details	2.9842
4	title_emb	1.4691
2	features_emb	0.7767
0	desc_emb	0.5023
3	price	0.0227

- Details fields collectively have the highest impact, confirming structured attributes are the strongest predictors
- Title embeddings rank second, showing product names contain significant classification signals
- Features embeddings contribute meaningfully but with lower importance
- Description embeddings provide moderate value
- Price has minimal impact, suggesting category is primarily determined by product attributes rather than cost

conclusions

Strengths

- Strong overall accuracy (~90%) with excellent top-k metrics (>97% for top-3)
- Several categories achieve near-perfect classification (Fashion, Digital Music)
- Structured product attributes (details fields) provide powerful predictive signals
- Title embeddings effectively capture product category information
- Model performs well even for some categories with limited samples

Weaknesses

- Significant cross-confusion between electronics-related categories
- Several low-support categories show extremely poor performance (F1 < 0.3)
- Imbalanced precision-recall trade-offs in some categories
- Price provides minimal predictive value despite being a key product attribute
- Performance inconsistency across categories with similar support sizes

Key Takeaways

- Product attributes and semantic content are more deterministic of category than price
- A minimum threshold of examples (~200-300) appears necessary for stable performance
- Electronics subcategories would benefit from more refined feature boundaries
- The model captures semantic patterns rather than simple keywords
- Confusion patterns largely follow intuitive category relationships

Next Steps

- Optimize classification thresholds for categories with imbalanced precision-recall
- Enhance feature engineering for frequently confused categories
- Consider hierarchical classification for related category groups
- Augment data for poorly-performing low-support categories
- Investigate performance on new products and category drift over time