In [24]:

```python
import pandas as pd
import networkx as nx

# Load the edge list
edges = pd.read_csv('musae_git_edges.csv')

# Load the target (labels)
targets = pd.read_csv('musae_git_target.csv')

# Create the graph
G = nx.from_pandas_edgelist(edges, source='id_1', target='id_2', create_using=nx.Graph())

# Add developer type attributes
for idx, row in targets.iterrows():
    node = row['id']
    if node in G.nodes:
        G.nodes[node]['developer_type'] = 'ML' if row['ml_target'] == 1 else 'Web'
```

In [25]:

```python
import networkx as nx
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import community  # Install with `pip install python-louvain`

# Set random seed for reproducibility
random.seed(42)

# ===========================
# 🔷 Question 1: Most Influential Developers (Degree Centrality)
# ===========================
degree_centrality = nx.degree_centrality(G)

# Optimize: Compute only for top 5% most connected nodes
degree_threshold = np.percentile(list(degree_centrality.values()), 95)
filtered_nodes = {node: centrality for node, centrality in degree_centrality.items() if centrality >= degree_thresho
```

```
top_10_influential = sorted(filtered_nodes.items(), key=lambda x: x[1], reverse=True)[:10]

influential_df = pd.DataFrame(top_10_influential, columns=['Node', 'Degree Centrality'])
influential_df['Developer Type'] = [G.nodes[node]['developer_type'] for node in influential_df['Node']]

# Round degree centrality for display
influential_df['Degree Centrality'] = influential_df['Degree Centrality'].round(3)

print("Top 10 Influential Developers:")
print(influential_df)

# Visualize top influential developers
plt.figure(figsize=(10, 6))
sns.barplot(x='Degree Centrality', y='Node', hue='Developer Type', palette=['#1f77b4', '#ff7f0e'], data=influential_(

# Customize the plot
plt.title('Top 10 Influential Developers by Degree Centrality', fontsize=14, pad=15)
plt.xlabel('Degree Centrality', fontsize=12)
plt.ylabel('Node ID', fontsize=12)
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.grid(True, axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# ===========================
# 🔷 Question 2: Community Detection (Web vs. ML Developers)
# ===========================
partition = community.best_partition(G)  # Louvain method
num_communities = len(set(partition.values()))
print(f"Number of communities detected: {num_communities}")

# Analyze developer types in the largest community
largest_community = max(set(partition.values()), key=list(partition.values()).count)
largest_nodes = [node for node in partition if partition[node] == largest_community]
community_types = [G.nodes[node]['developer_type'] for node in largest_nodes]

type_counts = pd.Series(community_types).value_counts()
plt.figure(figsize=(8, 6))
type_counts.plot(kind='bar', color=['#1f77b4', '#ff7f0e'])
plt.title('Developer Types in Largest Community', fontsize=14, pad=15)
plt.xlabel('Developer Type', fontsize=12)
plt.ylabel('Count', fontsize=12)
```

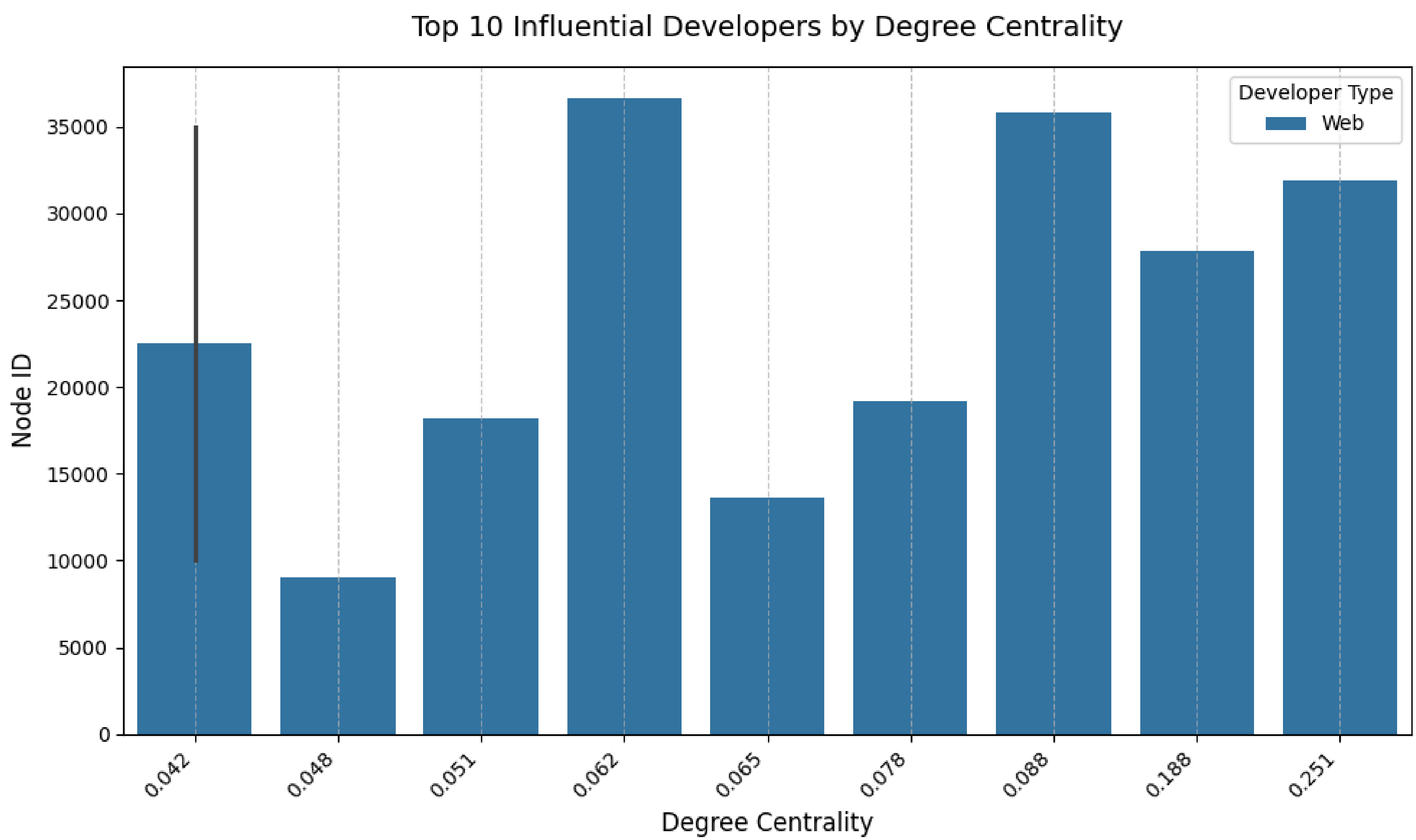
```
plt.xticks(rotation=0, fontsize=10)
plt.tight_layout()
plt.show()

# ==========================
# 🔷 Question 3: Degree Distribution by Developer Type
# ==========================
# Optimize: Use stratified sampling to balance Web and ML developers
web_nodes = [n for n in G.nodes if G.nodes[n]['developer_type'] == 'Web']
ml_nodes = [n for n in G.nodes if G.nodes[n]['developer_type'] == 'ML']
sample_size_per_type = 5000  # Sample 5000 from each type
sampled_web = random.sample(web_nodes, min(sample_size_per_type, len(web_nodes)))
sampled_ml = random.sample(ml_nodes, min(sample_size_per_type, len(ml_nodes)))
sampled_nodes = sampled_web + sampled_ml

web_degrees = [G.degree(n) for n in sampled_nodes if G.nodes[n]['developer_type'] == 'Web']
ml_degrees = [G.degree(n) for n in sampled_nodes if G.nodes[n]['developer_type'] == 'ML']

# Visualize degree distribution with a logarithmic scale
plt.figure(figsize=(10, 6))
sns.histplot(web_degrees, color='#1f77b4', label='Web Developers', kde=True, stat='density', bins=30, alpha=0.5)
sns.histplot(ml_degrees, color='#ff7f0e', label='ML Developers', kde=True, stat='density', bins=30, alpha=0.5)

# Use a logarithmic scale for the x-axis
plt.xscale('log')
plt.title('Degree Distribution by Developer Type (Log Scale)', fontsize=14, pad=15)
plt.xlabel('Degree (Log Scale)', fontsize=12)
plt.ylabel('Density', fontsize=12)
plt.legend(loc='upper right', fontsize=10)  # Explicitly set legend location
plt.tight_layout()
plt.show()

# ==========================
# 🔷 Question 4: Bridges Between Communities (Betweenness Centrality)
# ==========================
# Optimize: Use k=1000 for speed
betweenness = nx.betweenness_centrality(G, k=1000)
top_10_bridges = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]

bridges_df = pd.DataFrame(top_10_bridges, columns=['Node', 'Betweenness Centrality'])
bridges_df['Developer Type'] = [G.nodes[node]['developer_type'] for node in bridges_df['Node']]
```

```
print("Top 10 Bridges (Betweenness Centrality):")
print(bridges_df)

# ===========================
# 🔷 Question 5: Clustering Coefficient by Developer Type
# ===========================
# Optimize: Compute for a random subset of nodes
sampled_nodes = random.sample(list(G.nodes), min(5000, len(G.nodes)))
clustering = nx.clustering(G, nodes=sampled_nodes)

web_clustering = [clustering[n] for n in sampled_nodes if G.nodes[n]['developer_type'] == 'Web']
ml_clustering = [clustering[n] for n in sampled_nodes if G.nodes[n]['developer_type'] == 'ML']

plt.figure(figsize=(10, 6))
sns.boxplot(data=[web_clustering, ml_clustering], palette=['#1f77b4', '#ff7f0e'])
plt.xticks([0, 1], ['Web Developers', 'ML Developers'], fontsize=10)
plt.title('Clustering Coefficient by Developer Type', fontsize=14, pad=15)
plt.ylabel('Clustering Coefficient', fontsize=12)
plt.tight_layout()
plt.show()

# ===========================
# 🔷 Question 6: Predict Developer Type Using Network Properties
# ===========================
# Optimize: Limit dataset size for ML model training
sample_size = min(10000, len(G.nodes))
sampled_nodes = random.sample(list(G.nodes), sample_size)

features_df = pd.DataFrame({
    'Node': sampled_nodes,
    'Degree': [G.degree(n) for n in sampled_nodes],
    'Clustering': [clustering.get(n, 0) for n in sampled_nodes],
    'Betweenness': [betweenness.get(n, 0) for n in sampled_nodes]
})
features_df['Developer Type'] = [G.nodes[n]['developer_type'] for n in sampled_nodes]
features_df['Target'] = features_df['Developer Type'].map({'Web': 0, 'ML': 1})

# Prepare data for classification
X = features_df[['Degree', 'Clustering', 'Betweenness']]
y = features_df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

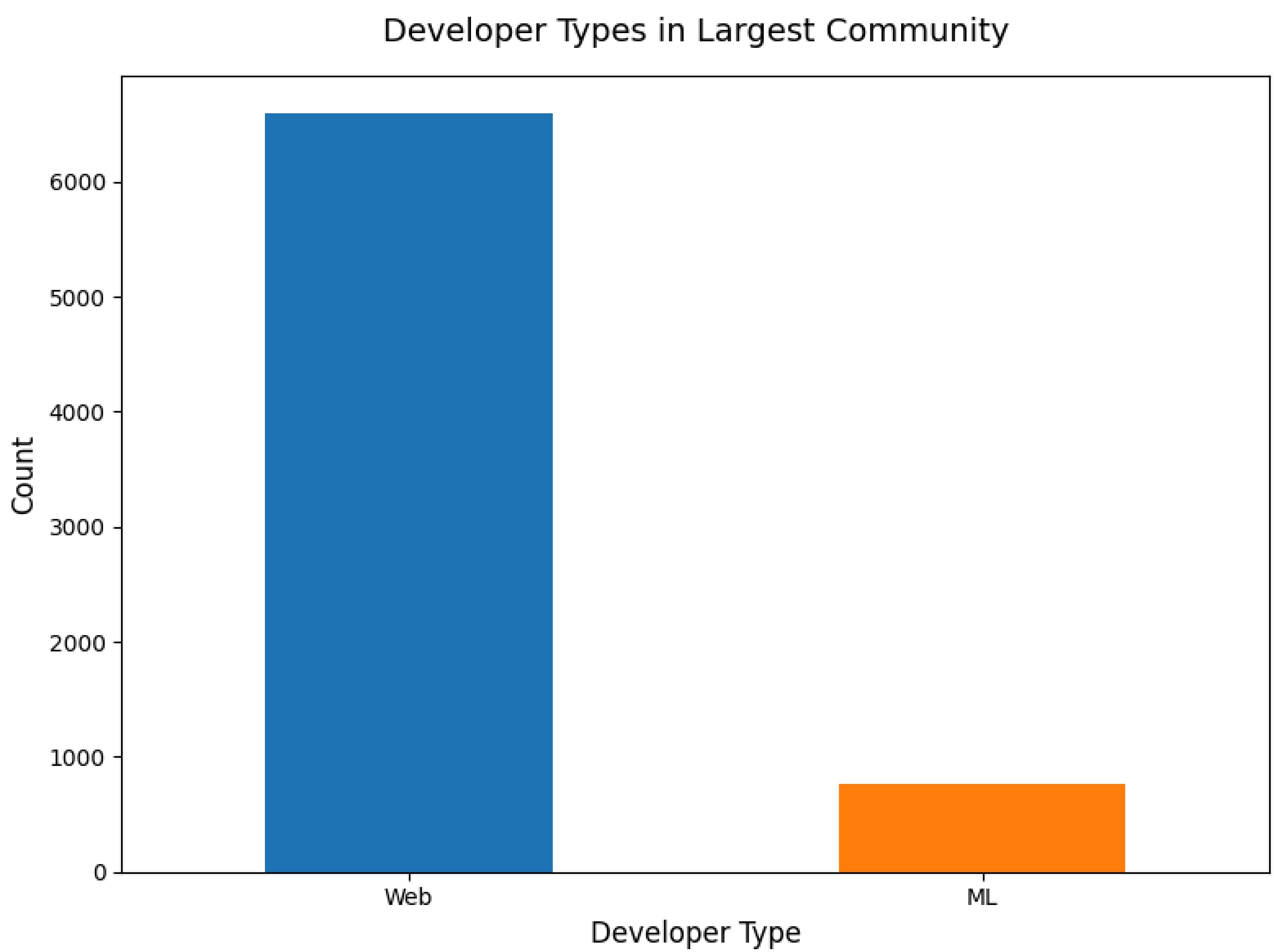
```python
# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy of predicting developer type: {accuracy:.2f}")
```

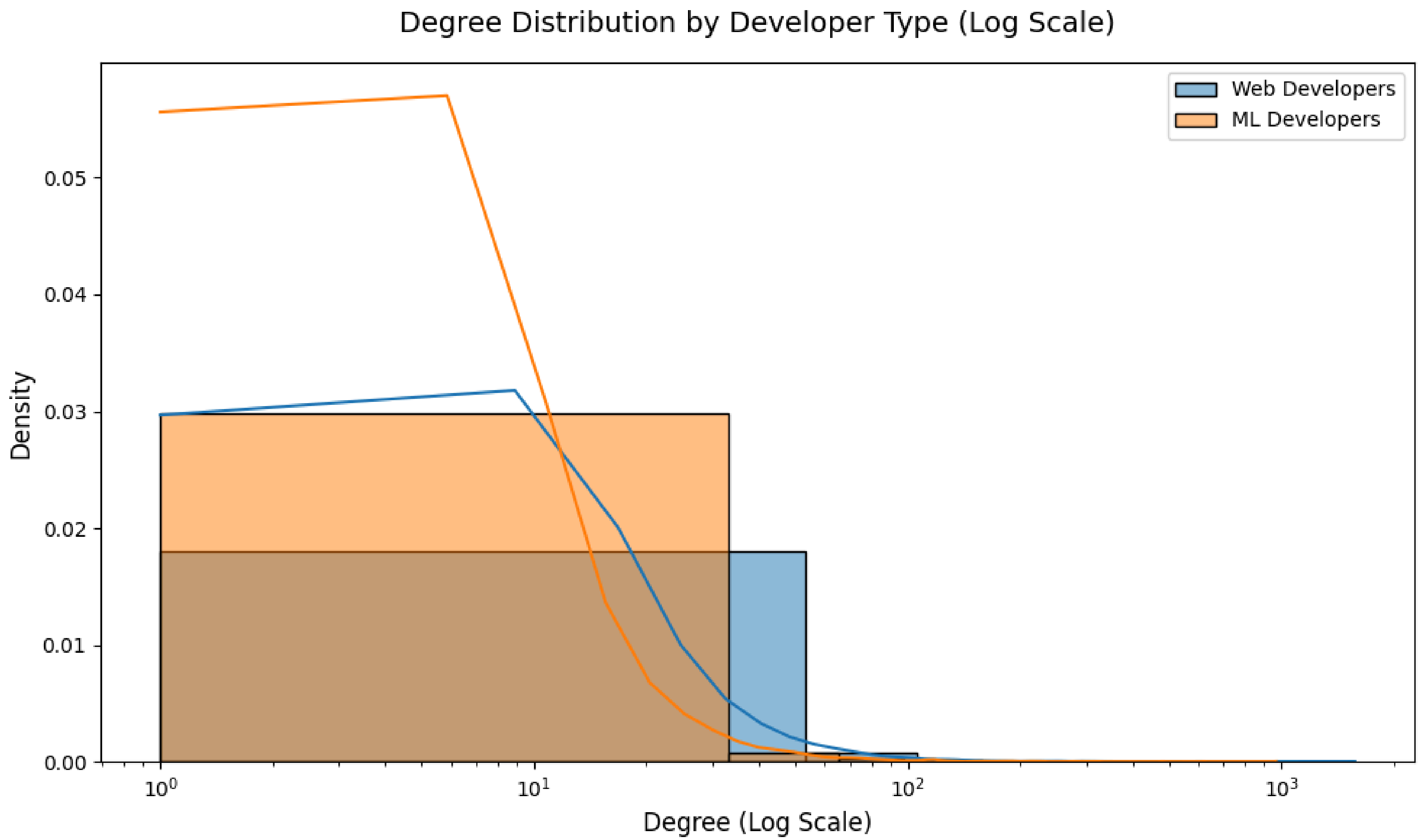
```
Top 10 Influential Developers:
    Node  Degree Centrality Developer Type
0  31890              0.251            Web
1  27803              0.188            Web
2  35773              0.088            Web
3  19222              0.078            Web
4  13638              0.065            Web
5  36652              0.062            Web
6  18163              0.051            Web
7   9051              0.048            Web
8  35008              0.042            Web
9  10001              0.042            Web
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_1360\928252992.py:36: UserWarning: The palette list has more values (2) than needed (1), which may not be intended.
  sns.barplot(x='Degree Centrality', y='Node', hue='Developer Type', palette=['#1f77b4', '#ff7f0e'], data=influential_df)
```




```
Number of communities detected: 34
```


Developer Types in Largest Community
Count
6000
5000
4000
3000
2000
1000
0
Web
ML
Developer Type



Top 10 Bridges (Betweenness Centrality):

	Node	Betweenness Centrality	Developer Type
0	31890	0.263204	Web
1	27803	0.240967	Web
2	19222	0.052186	Web
3	35773	0.046849	Web
4	13638	0.035719	Web
5	10001	0.028116	Web
6	36652	0.027381	Web
7	18163	0.024982	Web
8	5629	0.020634	Web
9	19253	0.019872	Web


```
Accuracy of predicting developer type: 0.75
```

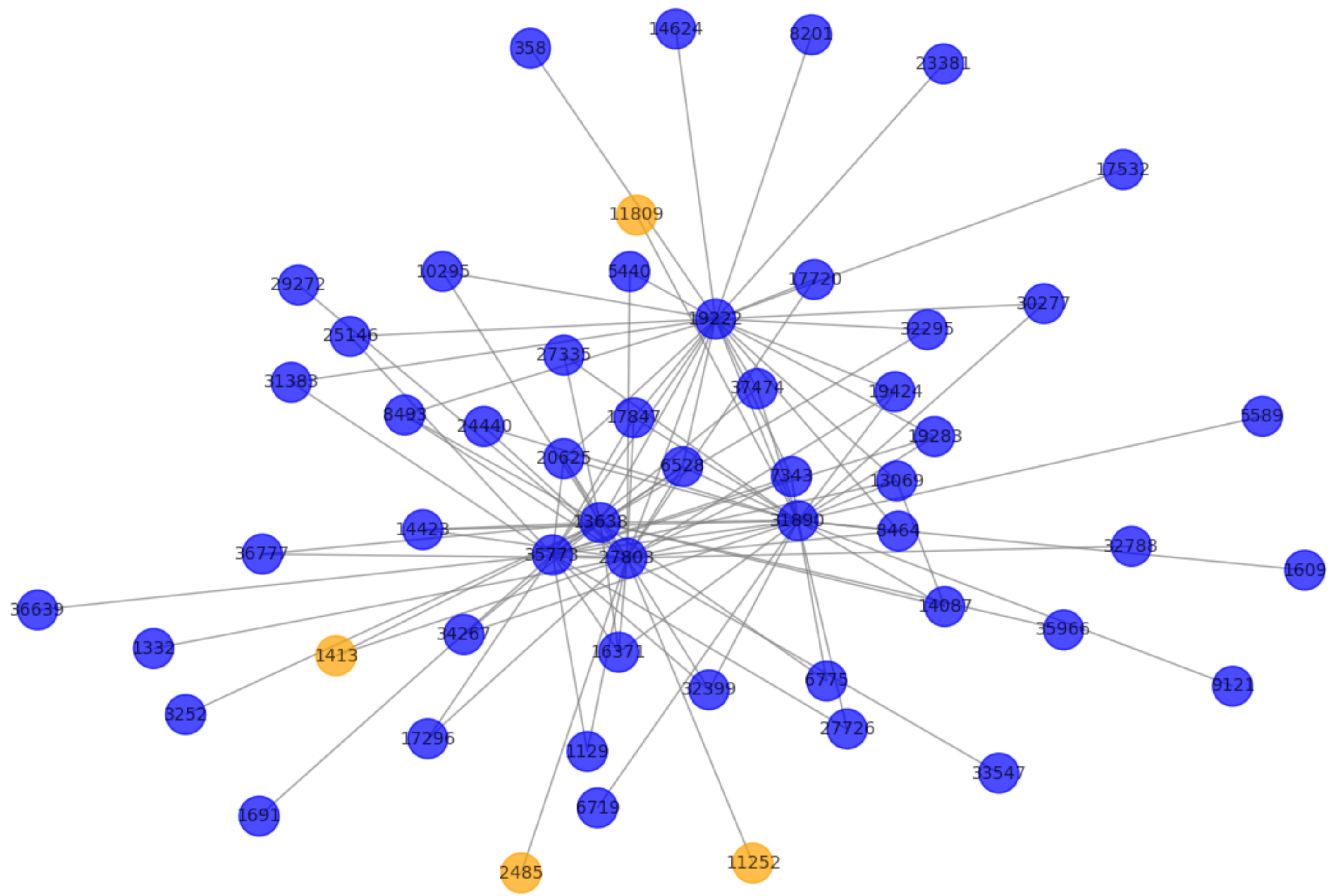
In [30]:
```
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
import random


# Set random seed for reproducibility
random.seed(42)

# ===========================
```

```
# Visualization 1: Subgraph of Top Influential Developers and Their Neighbors
# ===========================
# Reuse degree_centrality from previous cell
top_5_influential = sorted(degree_centrality.items(), key=lambda x: x[1], reverse=True)[:5]
top_nodes = [node for node, _ in top_5_influential]

# Create a subgraph with top 5 nodes and their immediate neighbors
subgraph_nodes = set(top_nodes)
for node in top_nodes:
    neighbors = list(G.neighbors(node))
    subgraph_nodes.update(random.sample(neighbors, min(10, len(neighbors))))

subgraph = G.subgraph(subgraph_nodes)

# Assign colors based on developer type
node_colors = ['blue' if subgraph.nodes[n]['developer_type'] == 'Web' else 'orange' for n in subgraph.nodes]

# Visualize the subgraph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(subgraph, k=0.3, iterations=50)
nx.draw(subgraph, pos, with_labels=True, node_color=node_colors, node_size=500, font_size=10, edge_color='gray', alph
plt.title('Subgraph of Top 5 Influential Developers and Their Neighbors\n(Blue: Web, Orange: ML)')
plt.show()

# ===========================
# Visualization 2: Community Structure (Largest Community Highlighted)
# ===========================
# Reuse partition from previous cell
largest_community = max(set(partition.values()), key=list(partition.values()).count)

# Create a subgraph of the largest community (limit to 50 nodes for visualization)
largest_nodes = [node for node in partition if partition[node] == largest_community]
largest_nodes_sample = random.sample(largest_nodes, min(50, len(largest_nodes)))
community_subgraph = G.subgraph(largest_nodes_sample)

# Assign colors based on developer type
community_colors = ['blue' if community_subgraph.nodes[n]['developer_type'] == 'Web' else 'orange' for n in community

# Visualize the largest community
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(community_subgraph, k=0.5, iterations=50)
nx.draw(community_subgraph, pos, with_labels=True, node_color=community_colors, node_size=500, font_size=10, edge_co
```

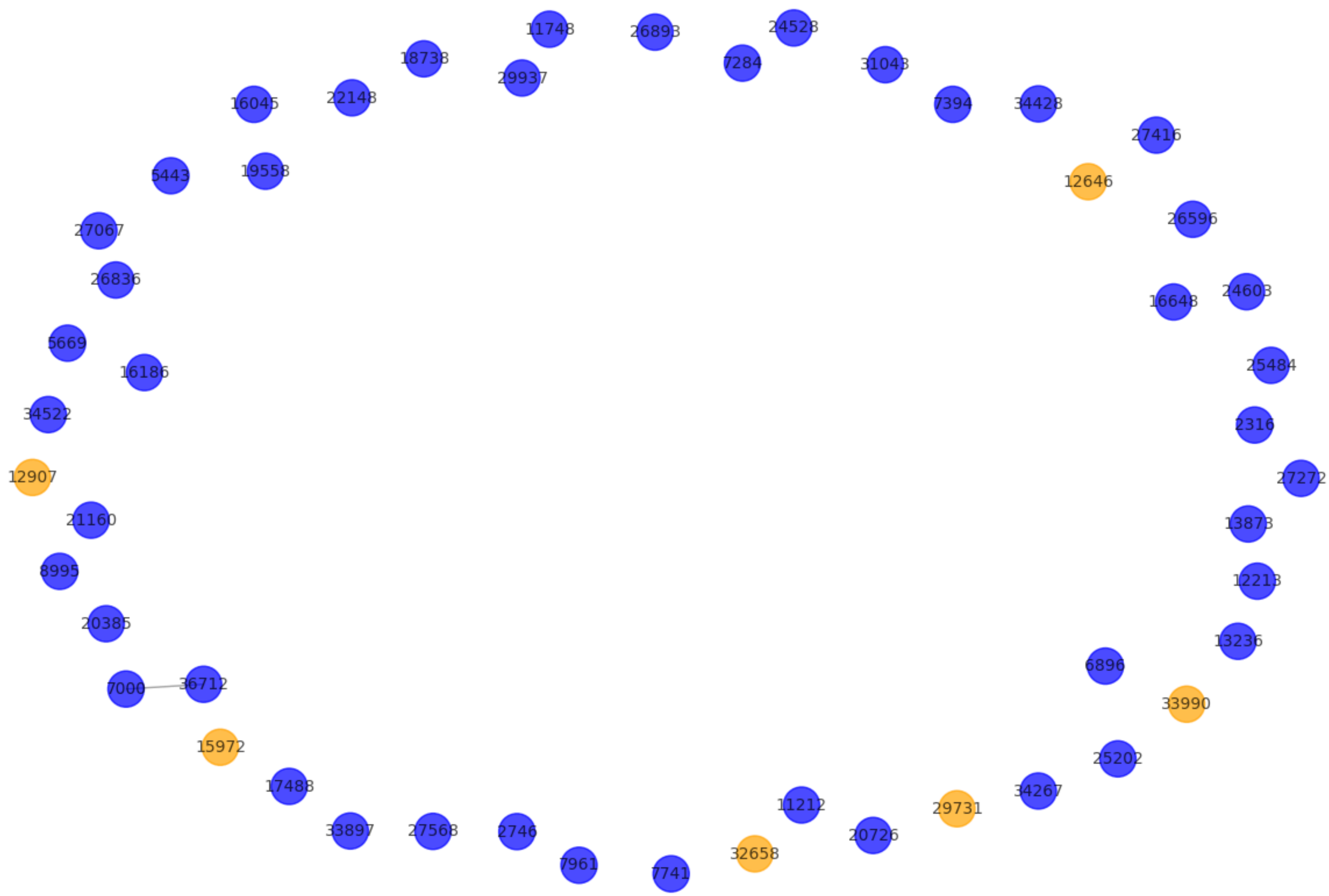
```python
plt.title('Largest Community Subgraph (50 Nodes Sample)\n(Blue: Web, Orange: ML)')
plt.show()

# ===========================
# Visualization 3: Bridges Between Communities (Betweenness Centrality)
# ===========================
# Reuse betweenness from previous cell
top_5_bridges = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:5]
bridge_nodes = [node for node, _ in top_5_bridges]

# Create a subgraph with top 5 bridge nodes and their neighbors
bridge_subgraph_nodes = set(bridge_nodes)
for node in bridge_nodes:
    neighbors = list(G.neighbors(node))
    bridge_subgraph_nodes.update(random.sample(neighbors, min(10, len(neighbors))))

bridge_subgraph = G.subgraph(bridge_subgraph_nodes)

# Assign colors based on developer type
bridge_colors = ['blue' if bridge_subgraph.nodes[n]['developer_type'] == 'Web' else 'orange' for n in bridge_subgraph

# Visualize the bridge subgraph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(bridge_subgraph, k=0.3, iterations=50)
nx.draw(bridge_subgraph, pos, with_labels=True, node_color=bridge_colors, node_size=500, font_size=10, edge_color='gr
plt.title('Subgraph of Top 5 Bridge Nodes (Betweenness Centrality) and Their Neighbors\n(Blue: Web, Orange: ML)')
plt.show()

# ===========================
# Visualization 4: Network Overview (Degree vs. Clustering Coefficient)
# ===========================
# Reuse clustering from previous cell, but extend to a new sample if needed
sample_size = 1000
sampled_nodes = random.sample(list(G.nodes), sample_size)

# Compute clustering for new sample if not already computed
clustering_extended = clustering.copy()
for node in sampled_nodes:
    if node not in clustering_extended:
        clustering_extended[node] = nx.clustering(G, node)

# Create a DataFrame for visualization
```

```python
scatter_df = pd.DataFrame({
    'Degree': [G.degree(n) for n in sampled_nodes],
    'Clustering': [clustering_extended[n] for n in sampled_nodes],
    'Developer Type': [G.nodes[n]['developer_type'] for n in sampled_nodes]
})

# Scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Degree', y='Clustering', hue='Developer Type', size='Degree', data=scatter_df, alpha=0.6)
plt.title('Degree vs. Clustering Coefficient (Sample of 1000 Nodes)')
plt.xlabel('Degree')
plt.ylabel('Clustering Coefficient')
plt.legend()
plt.show()
```

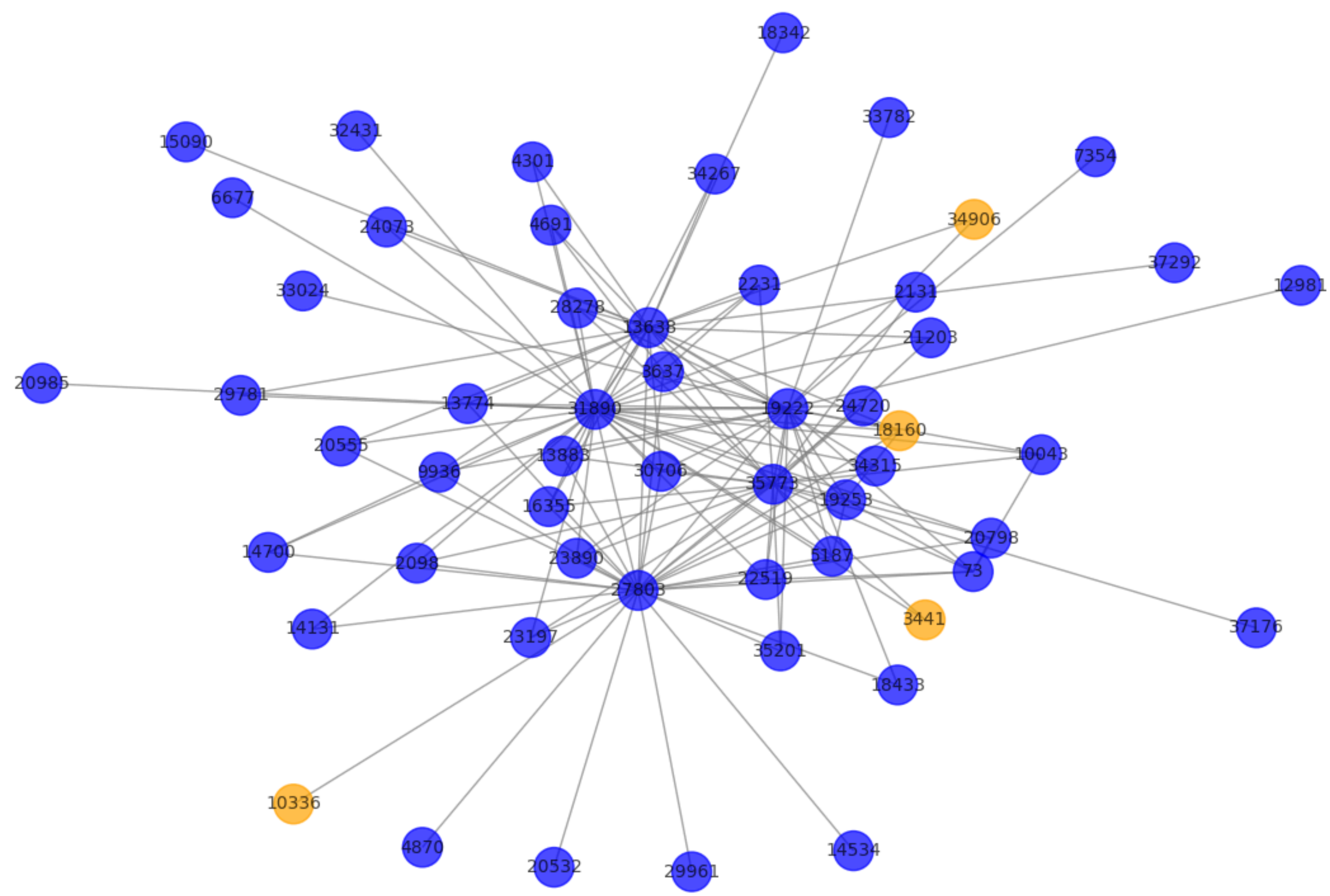
Subgraph of Top 5 Influential Developers and Their Neighbors
(Blue: Web, Orange: ML)

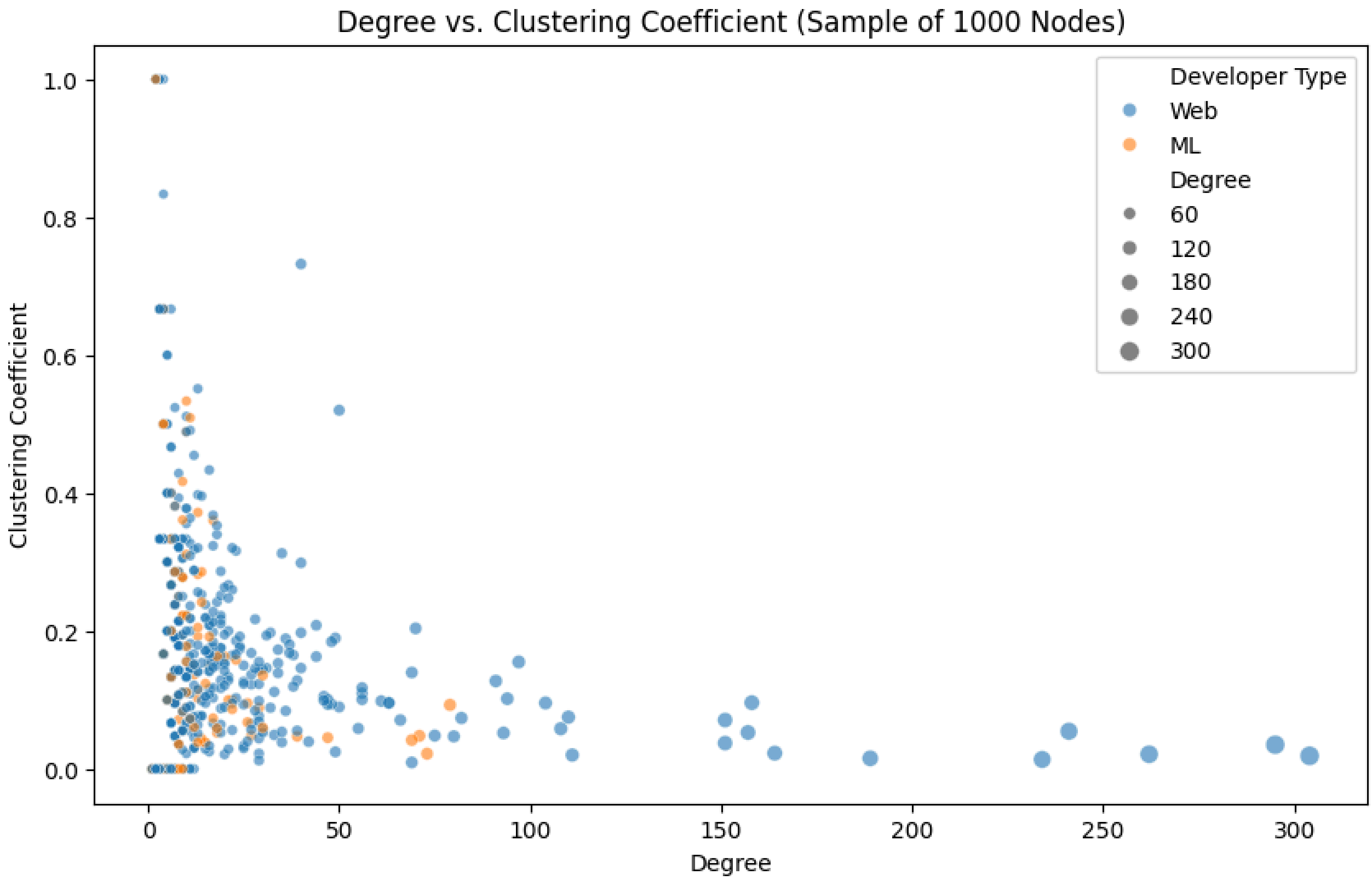


Largest Community Subgraph (50 Nodes Sample)
(Blue: Web, Orange: ML)



Subgraph of Top 5 Bridge Nodes (Betweenness Centrality) and Their Neighbors
(Blue: Web, Orange: ML)




Degree vs. Clustering Coefficient (Sample of 1000 Nodes)
Clustering Coefficient
Degree
Developer Type
Web
ML
Degree
60
120
180
240
300
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300