In [6]:
```
import os
import torch
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
from PIL import Image
import torch.nn as nn
import torch.optim as optim
import random
import pandas as pd
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve,
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import numpy as np
```

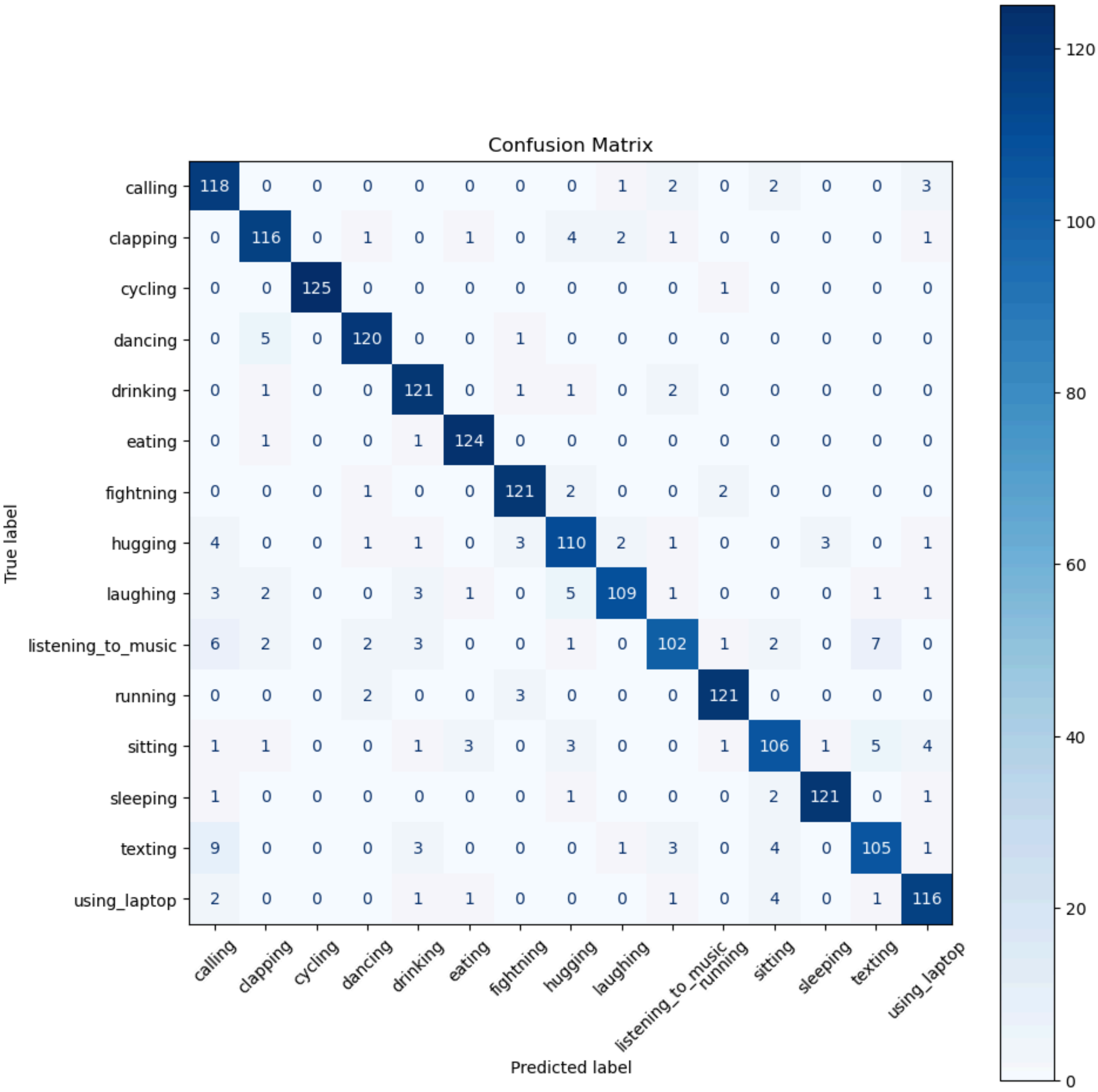
In [7]:
```
# classify an image with the given model path
def classify_image(image_path, model, device):
    class_names = [
        'calling',
        'clapping',
        'cycling',
        'dancing',
        'drinking',
        'eating',
        'fightning',
        'hugging',
        'laughing',
        'listening_to_music',
        'running',
        'sitting',
        'sleeping',
        'texting',
        'using_laptop'
    ]

    transform = transforms.Compose([
        transforms.Resize((256, 256)),
        transforms.ToTensor(),
        transforms.Normalize((0.5729, 0.5379, 0.5069), (0.3056, 0.3022, 0.3096))
    ])

    image = Image.open(f'./data2/test/{image_path}').convert('RGB')
    image = transform(image).unsqueeze(0).to(device)
    model.eval()

    with torch.no_grad():
        outputs = model(image)
        probs = torch.softmax(outputs, dim=1).cpu().numpy()[0]
        _, predicted = torch.max(outputs, 1)

    return predicted.item(), probs
```

In [8]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load the model
model_path = "models/resnet/resnet_model.pth"  # Replace with your actual model
model = models.resnet152()
model.fc = nn.Linear(model.fc.in_features, 15)
model.load_state_dict(torch.load("models/resnet/resnet_model.pth"
```

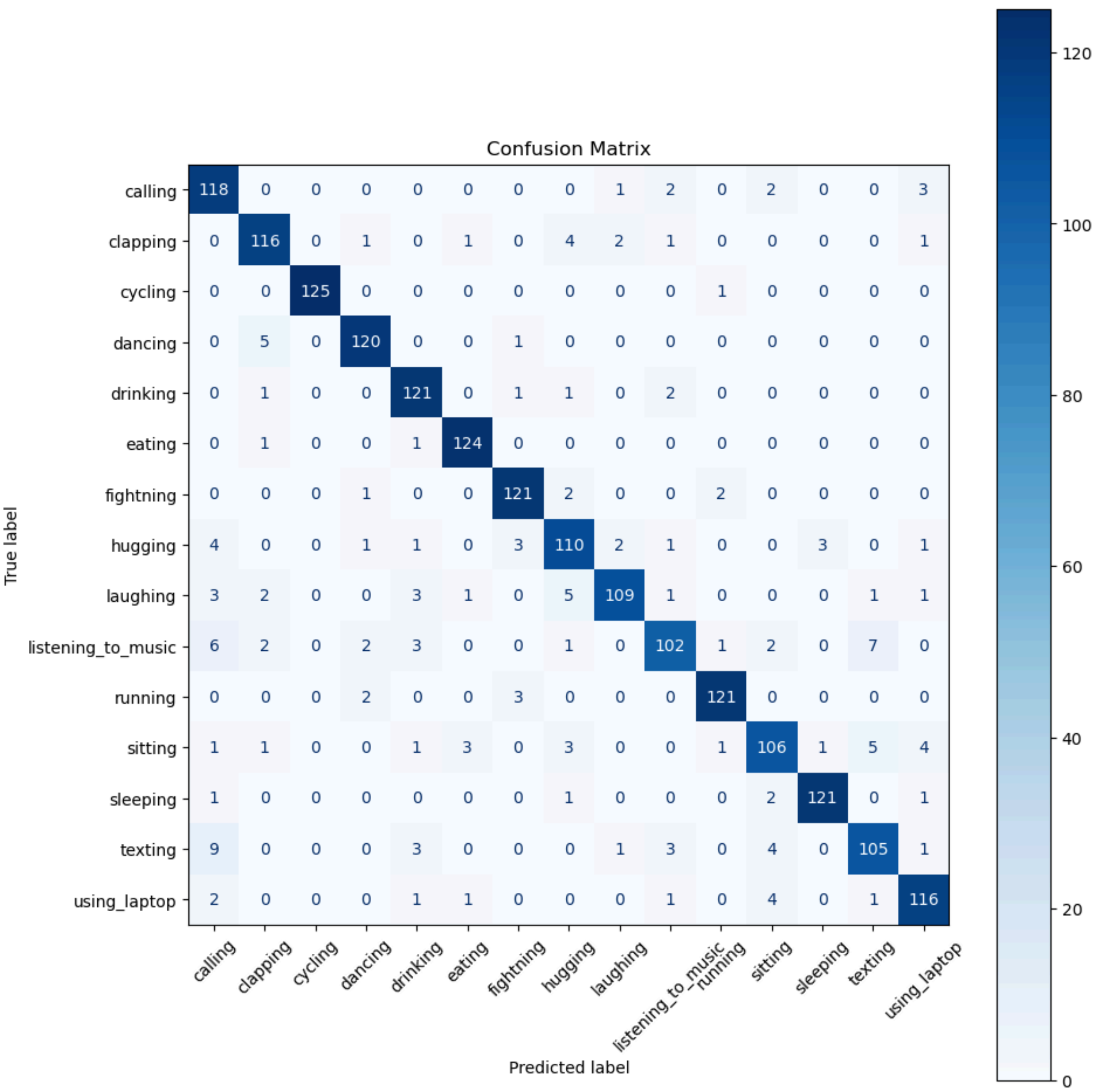
```
                                 ,map_location=torch.device("cpu")
))
model.to(device)

# Read the CSV file
# Using absolute path similar to the model path
test_data = pd.read_csv("data2/test/test_labels.csv")
#test_data = test_data.iloc[:50]
print(f"Test dataset contains {len(test_data)} images")

# Assuming first column is filename and third is the true label
# Adjust these if your CSV has different structure
filename_col = 0
folder_col = 1
label_col = 2

# Process images and compute accuracy
correct = 0
total = 0
all_preds = []
all_labels = []
predicted_probs = []

for index, row in test_data.iterrows():
    try:
        filename = row.iloc[filename_col]
        foldername = row.iloc[folder_col]
        filename = f'{foldername}/{filename}'
        true_label = row.iloc[label_col]

        predicted_label, probs = classify_image(filename, model, device)
        #print(f'Image: {filename}: ({predicted_label}, {true_label})')

        all_preds.append(predicted_label)
        all_labels.append(true_label)
        predicted_probs.append(probs)


        if predicted_label == true_label:
            correct += 1

        total += 1

        # Optional progress update
        if index % 50 == 0:
            print(f"Processed {index}/{len(test_data)} images")

    except Exception as e:
        print(f"Error processing {filename}: {e}")

# Calculate accuracy
accuracy = 100 * correct / total
print(f"\nAccuracy: {accuracy:.2f}% ({correct}/{total})")

class_names = [
        'calling',
        'clapping',
        'cycling',
        'dancing',
        'drinking',
```

```
        'eating',
        'fightning',
        'hugging',
        'laughing',
        'listening_to_music',
        'running',
        'sitting',
        'sleeping',
        'texting',
        'using_laptop'
    ]

cm = confusion_matrix(all_labels, all_preds)
fig, ax = plt.subplots(figsize=(10, 10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(ax=ax, cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
```

```
Test dataset contains 1890 images
Processed 0/1890 images
Processed 50/1890 images
Processed 100/1890 images
Processed 150/1890 images
Processed 200/1890 images
Processed 250/1890 images
Processed 300/1890 images
Processed 350/1890 images
Processed 400/1890 images
Processed 450/1890 images
Processed 500/1890 images
Processed 550/1890 images
Processed 600/1890 images
Processed 650/1890 images
Processed 700/1890 images
Processed 750/1890 images
Processed 800/1890 images
Processed 850/1890 images
Processed 900/1890 images
Processed 950/1890 images
Processed 1000/1890 images
Processed 1050/1890 images
Processed 1100/1890 images
Processed 1150/1890 images
Processed 1200/1890 images
Processed 1250/1890 images
Processed 1300/1890 images
Processed 1350/1890 images
Processed 1400/1890 images
Processed 1450/1890 images
Processed 1500/1890 images
Processed 1550/1890 images
Processed 1600/1890 images
Processed 1650/1890 images
Processed 1700/1890 images
Processed 1750/1890 images
Processed 1800/1890 images
Processed 1850/1890 images

Accuracy: 91.80% (1735/1890)
```




```
In [20]: cm = confusion_matrix(all_labels, all_preds)
fig, ax = plt.subplots(figsize=(10, 10))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(ax=ax, cmap='Blues', xticks_rotation=45)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
plt.savefig("confusionMatrix.png", dpi=300)
```

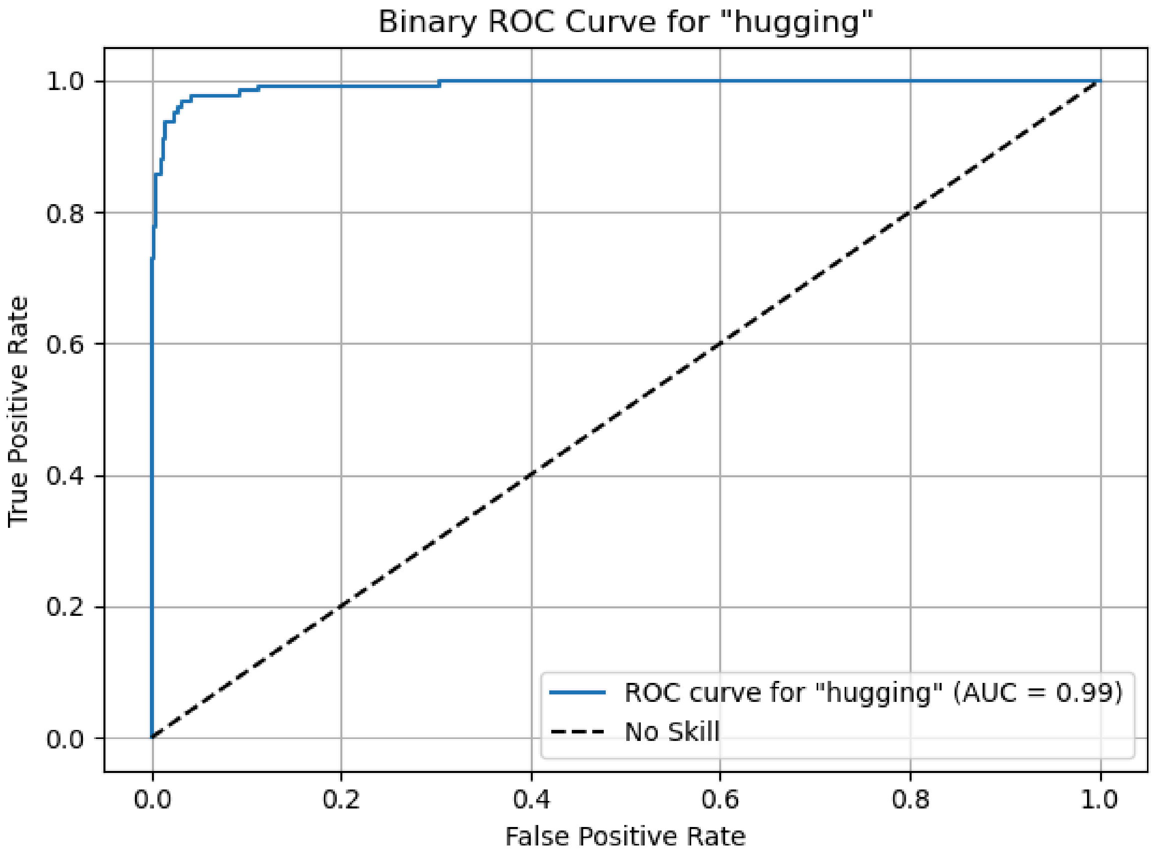



```
<Figure size 640x480 with 0 Axes>
```

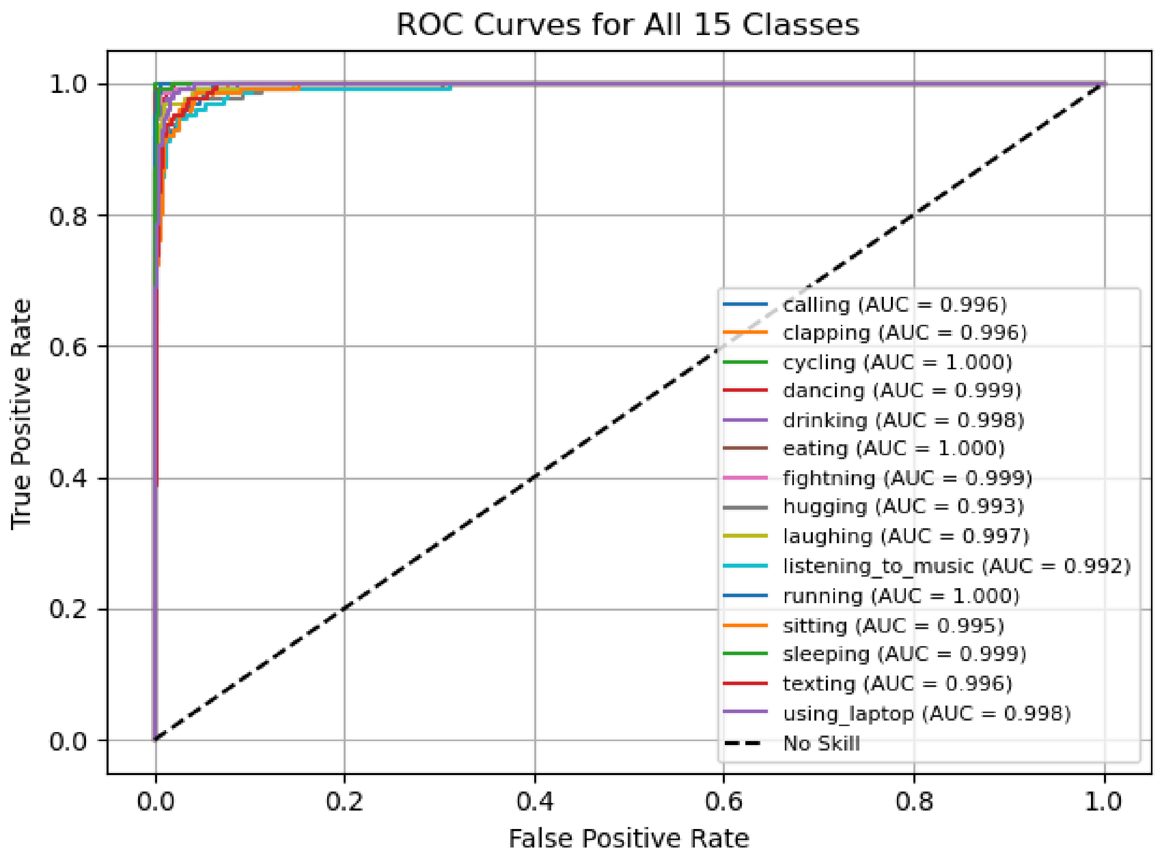
In [9]:

```
def plot_roc_curve(true_labels, predicted_probs, class_index, class_names):
    """
    Plots the ROC curve for a specific class index using true labels and predict
    """
    class_label = class_names[class_index]

    # Convert labels to binary (one-vs-rest)
    binary_labels = [1 if lbl == class_index else 0 for lbl in true_labels]
    class_probs = [prob[class_index] for prob in predicted_probs]

    fpr, tpr, _ = roc_curve(binary_labels, class_probs)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label=f'ROC curve for "{class_label}" (AUC = {roc_auc:.2f
    plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Binary ROC Curve for "{class_label}"')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
plot_roc_curve(all_labels, predicted_probs, class_index=7, class_names=class_nam
```



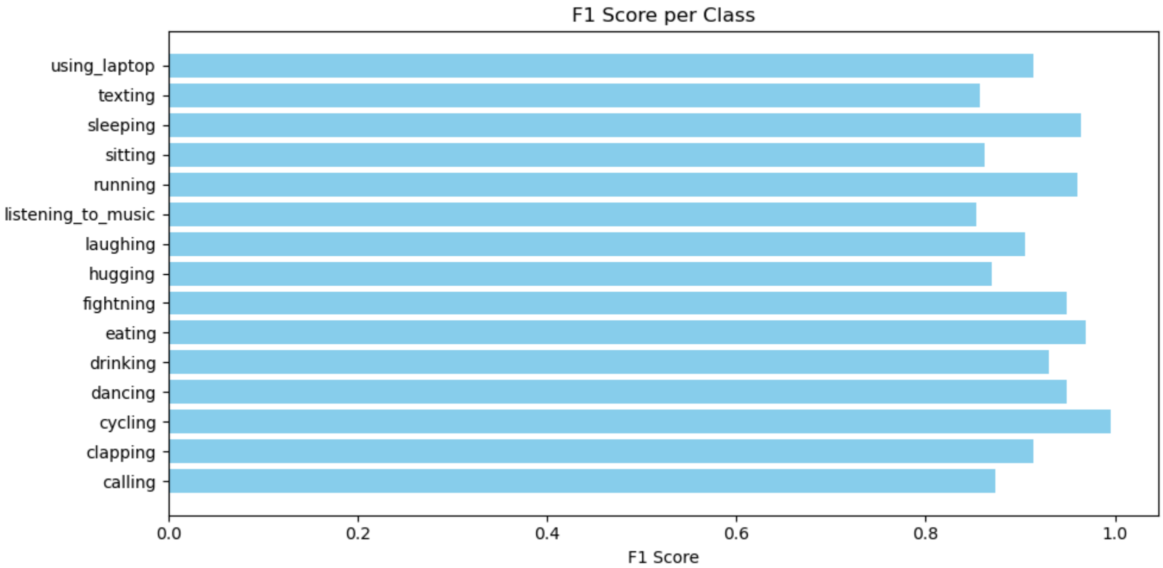
In [21]:
```
def plot_all_roc_curves(true_labels, predicted_probs, class_names):

    plt.figure()

    for class_index in range(len(class_names)):
        binary_labels = [1 if lbl == class_index else 0 for lbl in true_labels]
        class_probs = [prob[class_index] for prob in predicted_probs]

        if sum(binary_labels) == 0:
            continue  # Skip if no true samples for this class

        fpr, tpr, _ = roc_curve(binary_labels, class_probs)
        roc_auc = auc(fpr, tpr)

        plt.plot(fpr, tpr, label=f'{class_names[class_index]} (AUC = {roc_auc:.3

    plt.plot([0, 1], [0, 1], 'k--', label='No Skill')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curves for All 15 Classes')
    plt.legend(loc='lower right', fontsize=8)
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    plt.savefig("roc_all_classes.png", dpi=300)

plot_all_roc_curves(all_labels, predicted_probs, class_names)
```




```
<Figure size 640x480 with 0 Axes>
```

In [30]:
```
from sklearn.metrics import classification_report, f1_score
# Macro F1 (treat all classes equally)
f1_macro = f1_score(all_labels, all_preds, average='macro')

# Weighted F1 (accounts for class imbalance)
f1_weighted = f1_score(all_labels, all_preds, average='weighted')

# Per-class precision, recall, F1
report = classification_report(all_labels, all_preds, target_names=class_names)

print(f"F1 Score (macro): {f1_macro:.4f}")
print(f"F1 Score (weighted): {f1_weighted:.4f}")
print("\nDetailed Classification Report:\n")
print(report)

with open("classification_report.txt", "w") as f:
    f.write(f"F1 Score (macro): {f1_macro:.4f}\n")
    f.write(f"F1 Score (weighted): {f1_weighted:.4f}\n\n")
    f.write("Detailed Classification Report:\n\n")
    f.write(report)

report1 = classification_report(all_labels, all_preds, target_names=class_names,
labels = class_names
f1_scores = [report1[cls]["f1-score"] for cls in class_names]

plt.figure(figsize=(10, 5))
plt.barh(labels, f1_scores, color='skyblue')
plt.xlabel("F1 Score")
plt.title("F1 Score per Class")
plt.tight_layout()
plt.show()
plt.savefig("F1_Score.png", dpi=300)
```

```
F1 Score (macro): 0.9177
F1 Score (weighted): 0.9177

Detailed Classification Report:

                   precision    recall  f1-score   support

          calling       0.82      0.94      0.87       126
         clapping       0.91      0.92      0.91       126
          cycling       1.00      0.99      1.00       126
          dancing       0.94      0.95      0.95       126
         drinking       0.90      0.96      0.93       126
           eating       0.95      0.98      0.97       126
        fightning       0.94      0.96      0.95       126
          hugging       0.87      0.87      0.87       126
         laughing       0.95      0.87      0.90       126
listening_to_music       0.90      0.81      0.85       126
          running       0.96      0.96      0.96       126
          sitting       0.88      0.84      0.86       126
         sleeping       0.97      0.96      0.96       126
          texting       0.88      0.83      0.86       126
     using_laptop       0.91      0.92      0.91       126

         accuracy                           0.92      1890
        macro avg       0.92      0.92      0.92      1890
     weighted avg       0.92      0.92      0.92      1890
```



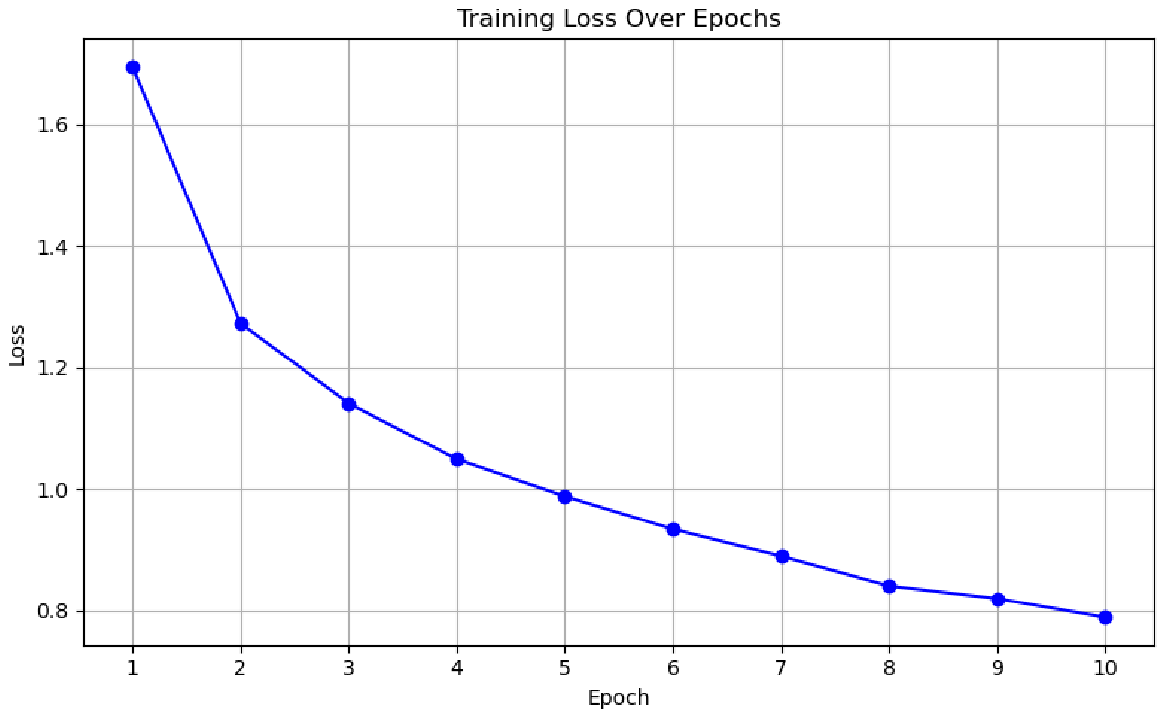

```
<Figure size 640x480 with 0 Axes>
```

In [25]:
```python
from collections import defaultdict

# Initialize counters
per_class_correct = defaultdict(int)
per_class_total = defaultdict(int)

# Loop through all predictions
for true, pred in zip(all_labels, all_preds):
    per_class_total[true] += 1
    if true == pred:
        per_class_correct[true] += 1

# Print results using class names
print("Correctly Classified Samples per Class:\n")
```

```
for idx, class_name in enumerate(class_names):
    correct = per_class_correct[idx]
    total = per_class_total[idx]
    print(f"{class_name:<20} {correct}/{total} ({(correct/total*100):.2f}%)")


with open("classification_results.txt", "w") as f:
    f.write("Correctly Classified Samples per Class:\n\n")
    for idx, class_name in enumerate(class_names):
        correct = per_class_correct[idx]
        total = per_class_total[idx]
        accuracy = (correct / total * 100) if total > 0 else 0.0
        f.write(f"{class_name:<20} {correct}/{total} ({accuracy:.2f}%)\n")
```

```
Correctly Classified Samples per Class:

calling              118/126 (93.65%)
clapping             116/126 (92.06%)
cycling              125/126 (99.21%)
dancing              120/126 (95.24%)
drinking             121/126 (96.03%)
eating               124/126 (98.41%)
fightning            121/126 (96.03%)
hugging              110/126 (87.30%)
laughing             109/126 (86.51%)
listening_to_music   102/126 (80.95%)
running              121/126 (96.03%)
sitting              106/126 (84.13%)
sleeping             121/126 (96.03%)
texting              105/126 (83.33%)
using_laptop         116/126 (92.06%)
```

In [15]:
```
losses = []
with open("loss.txt", "r") as f:
    for line in f:
        if "Loss" in line:
            parts = line.strip().split("Loss:")
            if len(parts) == 2:
                loss_value = float(parts[1].strip())
                losses.append(loss_value)

epochs = list(range(1, len(losses) + 1))


plt.figure(figsize=(8, 5))
plt.plot(epochs, losses, marker='o', linestyle='-', color='blue')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss Over Epochs")
plt.grid(True)
plt.xticks(epochs)
plt.tight_layout()
plt.show()
```



In [31]:
```
train_loss = losses
plt.plot(train_loss, label="Train Loss")
#plt.plot(val_loss, label="Val Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True)
plt.show()
```