In [1]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib as plt

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input di

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when yo
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current sessio
```

```
/kaggle/input/glycoiq2/Glyco_20250426_090045.jpg
/kaggle/input/glycoiq2/Glyco_20250426_121201.jpg
/kaggle/input/glycoiq2/Glyco_20250426_085243.jpg
/kaggle/input/glycoiq2/Glyco_20250426_095555.jpg
/kaggle/input/glycoiq2/Glyco_20250426_091307.jpg
/kaggle/input/glycoiq2/Glyco_20250426_095934.jpg
/kaggle/input/glycoiq2/Glyco_20250426_092111.jpg
/kaggle/input/glycoiq2/Glyco_20250426_121953.jpg
/kaggle/input/glycoiq2/data_log.csv
/kaggle/input/glycoiq4/Glyco_20250426_133243.jpg
/kaggle/input/glycoiq4/Glyco_20250426_090045.jpg
/kaggle/input/glycoiq4/Glyco_20250426_173708.jpg
/kaggle/input/glycoiq4/Glyco_20250426_121201.jpg
/kaggle/input/glycoiq4/Glyco_20250426_174252.jpg
/kaggle/input/glycoiq4/Glyco_20250426_174721.jpg
/kaggle/input/glycoiq4/Glyco_20250426_085243.jpg
/kaggle/input/glycoiq4/Glyco_20250426_095555.jpg
/kaggle/input/glycoiq4/data_collector.py
/kaggle/input/glycoiq4/Glyco_20250426_172923.jpg
/kaggle/input/glycoiq4/Glyco_20250426_091307.jpg
/kaggle/input/glycoiq4/Glyco_20250426_095934.jpg
/kaggle/input/glycoiq4/Glyco_20250426_092111.jpg
/kaggle/input/glycoiq4/Glyco_20250426_121953.jpg
/kaggle/input/glycoiq4/data_log.csv
/kaggle/input/4-4-sineth/Glyco_20250426_133243.jpg
```

In [2]:
```python
import os
import cv2
import numpy as np
import pandas as pd

# Paths (adjust as needed)
BASE_DIR = "/kaggle/input/glycoiq4"
IMAGE_DIR = BASE_DIR
CSV_LOG = os.path.join(BASE_DIR, "data_log.csv")

# Function to compute blue channel histogram
def compute_blue_histogram_features(image):
    img_uint8 = (image * 255).astype(np.uint8)
    hist_b = cv2.calcHist([img_uint8], [2], None, [256], [0, 256]).flatten()  # Blue channel
    return hist_b / hist_b.sum()  # Normalize to unit sum

# Load and preprocess data
print("Loading data...")
df = pd.read_csv(CSV_LOG)
image_names = df["Image Filename"].tolist()
glucose_levels = df["Blood Glucose Level (mmol/L)"].astype(float)  # Keep in mmol/L

# Process each image (one per volunteer)
images = []
```

```python
hist_features = []
valid_glucose_levels = []

for name, glucose in zip(image_names, glucose_levels):
    img_path = os.path.join(IMAGE_DIR, name)
    img = cv2.imread(img_path)
    if img is None:
        print(f"Warning: Image {img_path} not found.")
        continue
    # Resize to 640x480 (paper's resolution)
    img = cv2.resize(img, (640, 480))
    # Convert BGR to RGB
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Normalize to [0,1]
    img_normalized = img / 255.0
    # Compute blue channel histogram
    hist = compute_blue_histogram_features(img_normalized)
    images.append(img_normalized)
    hist_features.append(hist)
    valid_glucose_levels.append(glucose)

# Convert to numpy arrays
images = np.array(images)
hist_features = np.array(hist_features)
valid_glucose_levels = np.array(valid_glucose_levels)

print(f"Processed {len(hist_features)} volunteers with {hist_features.shape[1]} histogram features each.")
```

```
Loading data...
Processed 13 volunteers with 256 histogram features each.
```

In [3]:

```python
print(valid_glucose_levels)
```

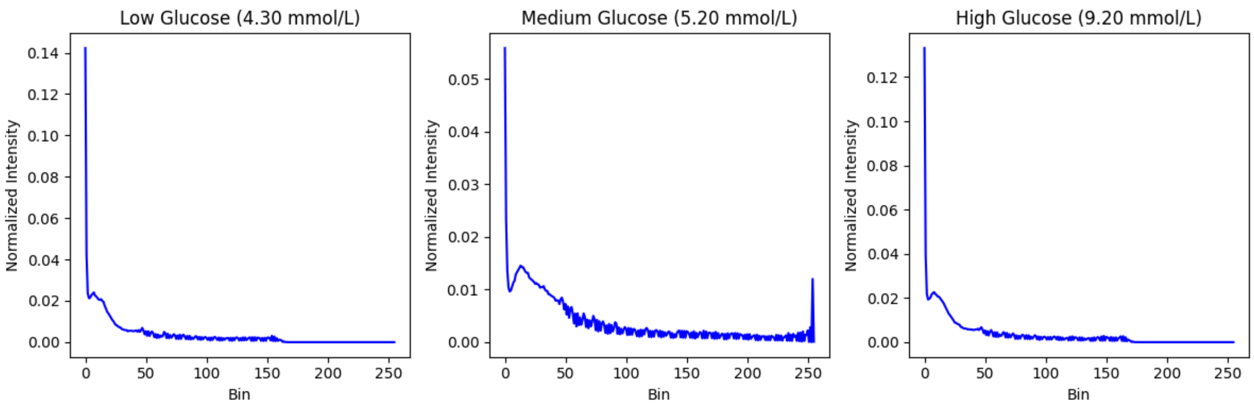
```
[4.3 4.8 4.7 6.8 9.2 5.3 5.3 6.1 4.4 6.8 4.7 5.2 5.2]
```

In [4]:

```python
import matplotlib.pyplot as plt
import numpy as np

# Visualize blue channel histograms
def plot_histograms(hist_features, glucose_levels, num_samples=3):
    """
    Plot blue channel histograms for low, medium, and high glucose levels.

    Args:
        hist_features: Numpy array of shape (N, 256) containing histogram features.
        glucose_levels: Numpy array of shape (N,) containing glucose levels in mmol/L.
        num_samples: Number of samples to plot (default: 3).
    """
    # Sort by glucose level and select low, medium, high
    indices = np.argsort(glucose_levels)
    low_idx = indices[0]
    mid_idx = indices[len(indices)//2]
    high_idx = indices[-1]

    plt.figure(figsize=(12, 4))

    # Low glucose
    plt.subplot(1, 3, 1)
    plt.plot(hist_features[low_idx], color='blue')
    plt.title(f'Low Glucose ({glucose_levels[low_idx]:.2f} mmol/L)')
    plt.xlabel('Bin')
    plt.ylabel('Normalized Intensity')

    # Medium glucose
    plt.subplot(1, 3, 2)
    plt.plot(hist_features[mid_idx], color='blue')
    plt.title(f'Medium Glucose ({glucose_levels[mid_idx]:.2f} mmol/L)')
    plt.xlabel('Bin')
    plt.ylabel('Normalized Intensity')

    # High glucose
```

```
    plt.subplot(1, 3, 3)
    plt.plot(hist_features[high_idx], color='blue')
    plt.title(f'High Glucose ({glucose_levels[high_idx]:.2f} mmol/L)')
    plt.xlabel('Bin')
    plt.ylabel('Normalized Intensity')

    plt.tight_layout()
    plt.show()

print("Visualizing histograms...")
plot_histograms(hist_features, valid_glucose_levels)
```

```
Visualizing histograms...
```



In [6]:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Create ANN model
def create_ann_model():
    inputs = Input(shape=(256,), name='hist_input')  # 256 blue histogram bins
    x = Dense(1024, activation='relu')(inputs)  # First hidden layer
    x = Dropout(0.2)(x)
    x = Dense(1024, activation='relu')(x)  # Second hidden layer
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation=None)(x)  # Regression output
    model = Model(inputs, outputs)
    return model

print("Building and training model...")
model = create_ann_model()
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

# Train model
history = model.fit(
    hist_features,
    valid_glucose_levels,
    epochs=100,
    batch_size=50,
    verbose=1
)
```

```
2025-04-26 12:34:32.191014: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1745670872.430234      80 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1745670872.496456      80 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
Building and training model...
```

```
I0000 00:00:1745670885.785949      80 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1745670885.786627      80 gpu_device.cc:2022] Created device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
Epoch 1/100
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1745670888.686181     138 service.cc:148] XLA service 0x7a00d400bce0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
I0000 00:00:1745670888.686880     138 service.cc:156]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5
I0000 00:00:1745670888.686900     138 service.cc:156]   StreamExecutor device (1): Tesla T4, Compute Capability 7.5
I0000 00:00:1745670888.904710     138 cuda_dnn.cc:529] Loaded cuDNN version 90300
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - loss: 33.1335 - mae: 5.6074
Epoch 2/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 32.1091 - mae: 5.5152
Epoch 3/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 31.0582 - mae: 5.4191
Epoch 4/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 29.7313 - mae: 5.2932
Epoch 5/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 28.2110 - mae: 5.1504
Epoch 6/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 26.2571 - mae: 4.9565
Epoch 7/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 23.7726 - mae: 4.6964
Epoch 8/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 21.1829 - mae: 4.4073
Epoch 9/100
I0000 00:00:1745670890.379875     138 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 18.0579 - mae: 4.0455
Epoch 10/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 14.6940 - mae: 3.6000
Epoch 11/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 11.2079 - mae: 3.0825
Epoch 12/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 7.6991 - mae: 2.4621
Epoch 13/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 5.0951 - mae: 1.8610
Epoch 14/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 2.9669 - mae: 1.1744
Epoch 15/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.8943 - mae: 0.9271
Epoch 16/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.1869 - mae: 1.2942
Epoch 17/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 3.4700 - mae: 1.6757
Epoch 18/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 5.4195 - mae: 2.1185
Epoch 19/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 6.3789 - mae: 2.3483
Epoch 20/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 5.8114 - mae: 2.2572
Epoch 21/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 5.3779 - mae: 2.1473
Epoch 22/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 4.1940 - mae: 1.8557
Epoch 23/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 3.0819 - mae: 1.5943
Epoch 24/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.0945 - mae: 1.1902
Epoch 25/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6195 - mae: 1.0277
Epoch 26/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.8147 - mae: 0.9131
Epoch 27/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.9386 - mae: 0.8964
Epoch 28/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.1569 - mae: 0.9816
Epoch 29/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.4371 - mae: 1.0674
Epoch 30/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.4679 - mae: 1.0989
Epoch 31/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.8942 - mae: 1.1474
Epoch 32/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 3.0130 - mae: 1.2022
Epoch 33/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.7964 - mae: 1.1522
Epoch 34/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 2.6922 - mae: 1.1541
Epoch 35/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.5904 - mae: 1.1459
Epoch 36/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 2.2119 - mae: 1.0581
Epoch 37/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.9943 - mae: 1.0036
Epoch 38/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7295 - mae: 0.8918
Epoch 39/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.8503 - mae: 0.9757
Epoch 40/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6609 - mae: 0.9696
Epoch 41/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6639 - mae: 1.1057
Epoch 42/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.9209 - mae: 1.1819
Epoch 43/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.8668 - mae: 1.1799
Epoch 44/100
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.8094 - mae: 1.1583
Epoch 45/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.9789 - mae: 1.1262
Epoch 46/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.8480 - mae: 1.1706
Epoch 47/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.9836 - mae: 1.1974
Epoch 48/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7036 - mae: 1.0463
Epoch 49/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7693 - mae: 1.0820
Epoch 50/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 1.5608 - mae: 0.9748
Epoch 51/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.9154 - mae: 1.0135
Epoch 52/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 1.5711 - mae: 0.8606
Epoch 53/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7825 - mae: 0.8935
Epoch 54/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.8259 - mae: 0.9431
Epoch 55/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.9660 - mae: 0.9424
Epoch 56/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5883 - mae: 0.8371
Epoch 57/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7647 - mae: 0.9433
Epoch 58/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.4832 - mae: 0.8605
Epoch 59/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7005 - mae: 0.8825
Epoch 60/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 1.5642 - mae: 0.8689
Epoch 61/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.8902 - mae: 1.0355
Epoch 62/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6767 - mae: 0.9743
Epoch 63/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5132 - mae: 0.8810
Epoch 64/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7474 - mae: 0.9486
Epoch 65/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7519 - mae: 1.0231
Epoch 66/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - loss: 1.6362 - mae: 0.9559
Epoch 67/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 1.4505 - mae: 0.9971
Epoch 68/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - loss: 1.5680 - mae: 0.9664
Epoch 69/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.4748 - mae: 0.9615
Epoch 70/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5513 - mae: 0.9502
Epoch 71/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6420 - mae: 1.0550
Epoch 72/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7122 - mae: 0.9537
Epoch 73/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6100 - mae: 0.9506
Epoch 74/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6015 - mae: 0.9204
Epoch 75/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5980 - mae: 0.9375
Epoch 76/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7671 - mae: 1.0045
Epoch 77/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.4664 - mae: 0.9064
Epoch 78/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.2721 - mae: 0.8432
Epoch 79/100
```

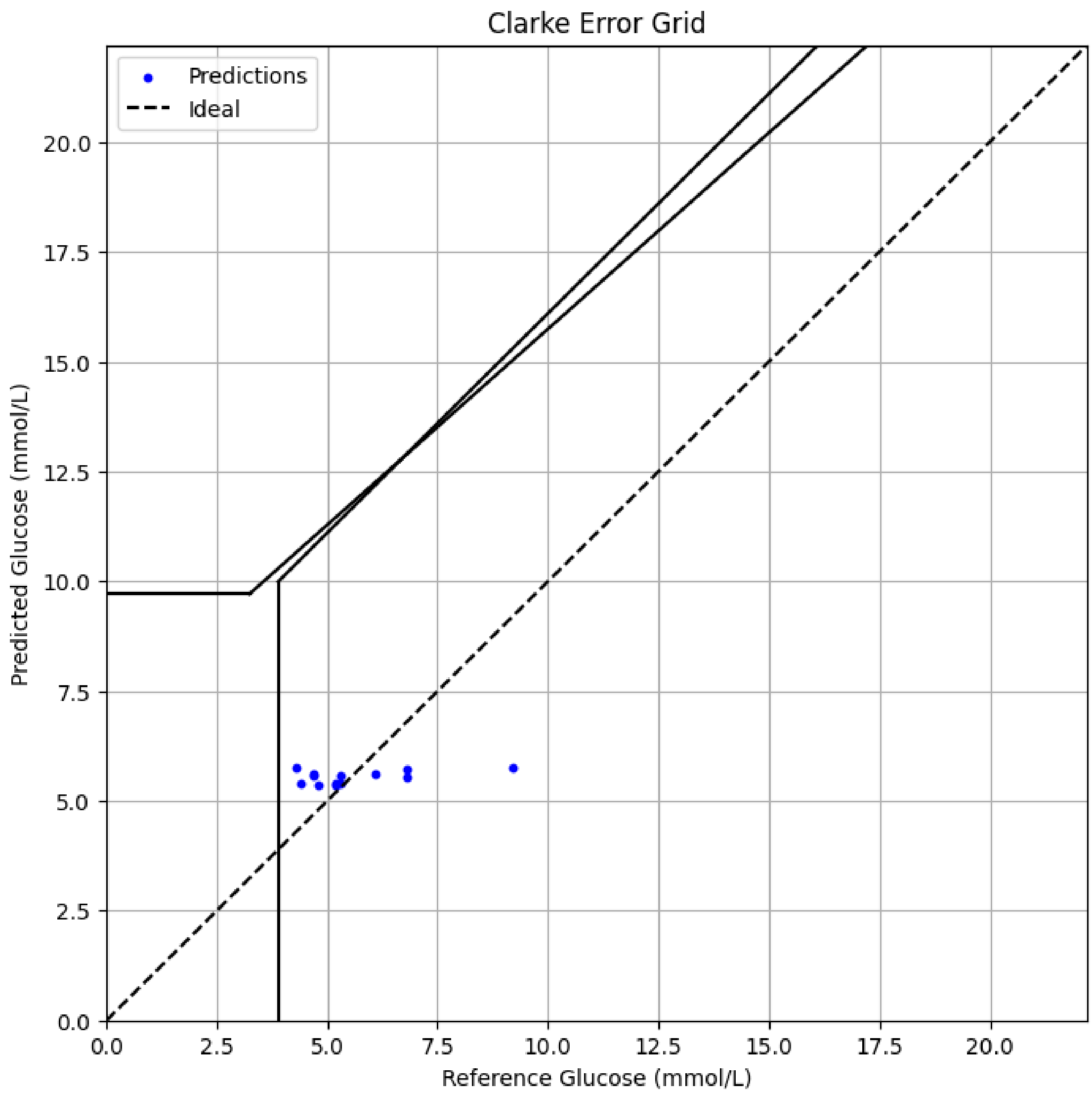
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.4522 - mae: 0.8525
Epoch 80/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5942 - mae: 0.9284
Epoch 81/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.2495 - mae: 0.7603
Epoch 82/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.6992 - mae: 0.9814
Epoch 83/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7675 - mae: 0.9722
Epoch 84/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.5119 - mae: 0.9267
Epoch 85/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7811 - mae: 1.0274
Epoch 86/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.7358 - mae: 0.9638
Epoch 87/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.6777 - mae: 0.9617
Epoch 88/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.3440 - mae: 0.8971
Epoch 89/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6068 - mae: 0.9619
Epoch 90/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6518 - mae: 0.9565
Epoch 91/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7554 - mae: 1.0099
Epoch 92/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.7005 - mae: 1.0007
Epoch 93/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5387 - mae: 0.9203
Epoch 94/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - loss: 1.5570 - mae: 0.9351
Epoch 95/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.6309 - mae: 0.9657
Epoch 96/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.4506 - mae: 0.9401
Epoch 97/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5991 - mae: 0.9199
Epoch 98/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.7475 - mae: 0.9599
Epoch 99/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - loss: 1.5726 - mae: 0.9096
Epoch 100/100
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 1.5919 - mae: 0.8954
```

In [8]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, r2_score

# Evaluate model
print("Making predictions...")
predictions = model.predict(hist_features).flatten()

# Compute MAE and R²
mae = mean_absolute_error(valid_glucose_levels, predictions)
r2 = r2_score(valid_glucose_levels, predictions)
print(f"Mean Absolute Error: {mae:.2f} mmol/L")
print(f"R² Score: {r2:.4f}")

# Clarke Error Grid
def plot_clarke_grid(y_true, y_pred):
    plt.figure(figsize=(8, 8))
    plt.scatter(y_true, y_pred, c='blue', s=10, label='Predictions')
    plt.plot([0, 22.2], [0, 22.2], 'k--', label='Ideal')  # 400 mg/dL ≈ 22.2 mmol/L
    # Zone A boundaries
    plt.plot([0, 3.24], [9.71, 9.71], 'k-')  # 175/3 ≈ 3.24 mmol/L, 175 ≈ 9.71 mmol/L
    plt.plot([3.24, 22.2], [9.71, 26.64], 'k-')  # 400*1.2 ≈ 26.64 mmol/L
    plt.plot([3.89, 3.89], [0, 9.99], 'k-')  # 70 ≈ 3.89 mmol/L, 180 ≈ 9.99 mmol/L
    plt.plot([3.89, 16.10], [9.99, 22.2], 'k-')  # 290 ≈ 16.10 mmol/L
    plt.xlim(0, 22.2)
    plt.ylim(0, 22.2)
```

```python
    plt.xlabel('Reference Glucose (mmol/L)')
    plt.ylabel('Predicted Glucose (mmol/L)')
    plt.title('Clarke Error Grid')
    plt.legend()
    plt.grid(True)
    plt.show()

# Compute Clarke Error Grid zones
def clarke_error_grid(y_true, y_pred):
    zones = {'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0}
    for ref, pred in zip(y_true, y_pred):
        if ref <= 3.89 and pred <= 3.89:  # 70 mg/dL ≈ 3.89 mmol/L
            zones['A'] += 1
        elif ref <= 3.89 and pred > 3.89 and pred <= 9.99:  # 180 mg/dL ≈ 9.99 mmol/L
            zones['D'] += 1
        elif ref > 13.32 and pred <= 3.89:  # 240 mg/dL ≈ 13.32 mmol/L
            zones['E'] += 1
        elif ref > 9.99 and pred <= 3.89:  # 180 mg/dL ≈ 9.99 mmol/L
            zones['D'] += 1
        elif ref > 3.89 and ref <= 9.99 and pred > 9.99 and pred <= 13.32:
            zones['C'] += 1
        elif ref > 9.99 and pred > 3.89 and pred <= 9.99:
            zones['D'] += 1
        elif abs(pred - ref) / ref <= 0.2:
            zones['A'] += 1
        else:
            zones['B'] += 1
    total = sum(zones.values())
    for zone in zones:
        zones[zone] = (zones[zone] / total) * 100
    return zones

print("Plotting Clarke Error Grid...")
plot_clarke_grid(valid_glucose_levels, predictions)
clarke_zones = clarke_error_grid(valid_glucose_levels, predictions)
print("Clarke Error Grid Zones (%):")
for zone, percentage in clarke_zones.items():
    print(f"Zone {zone}: {percentage:.2f}%")
```

```
Making predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Mean Absolute Error: 0.90 mmol/L
R² Score: 0.0923
Plotting Clarke Error Grid...
```




```
Clarke Error Grid Zones (%):
Zone A: 76.92%
Zone B: 23.08%
Zone C: 0.00%
Zone D: 0.00%
Zone E: 0.00%
```

In [9]:

```
print("Converting model to TFLite...")
tflite_path = "/kaggle/working/ann_model.tflite"
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with open(tflite_path, "wb") as f:
    f.write(tflite_model)
print(f"TFLite model saved to {tflite_path}")
```

```
Converting model to TFLite...
Saved artifact at '/tmp/tmpiqv88zeo'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 256), dtype=tf.float32, name='hist_input')
Output Type:
  TensorSpec(shape=(None, 1), dtype=tf.float32, name=None)
Captures:
  134147748129168: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134147748130512: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134147748133776: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134147748133392: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134147748134544: TensorSpec(shape=(), dtype=tf.resource, name=None)
  134147748133584: TensorSpec(shape=(), dtype=tf.resource, name=None)
TFLite model saved to /kaggle/working/ann_model.tflite
```

```
W0000 00:00:1745671036.594332      80 tf_tfl_flatbuffer_helpers.cc:365] Ignored output_format.
W0000 00:00:1745671036.594369      80 tf_tfl_flatbuffer_helpers.cc:368] Ignored drop_control_dependency.
I0000 00:00:1745671036.599450      80 mlir_graph_optimization_pass.cc:401] MLIR V1 optimization pass is not e
nabled
```

In [10]:

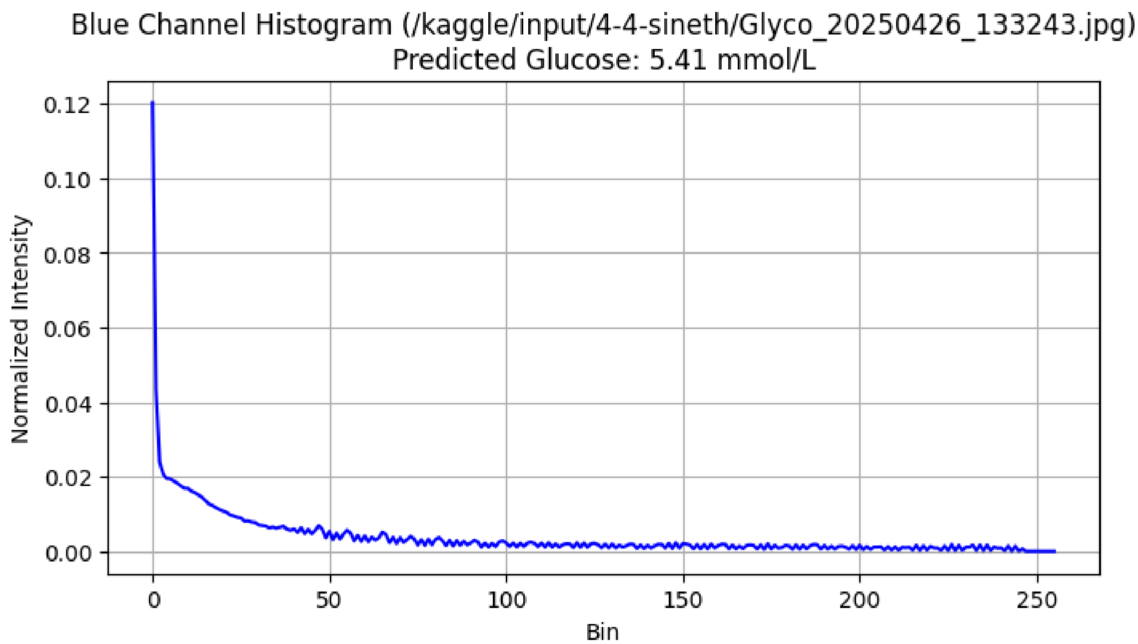
```python
# Assume compute_blue_histogram_features is defined elsewhere
def compute_blue_histogram_features(image):
    img_uint8 = (image * 255).astype(np.uint8)
    hist_b = cv2.calcHist([img_uint8], [2], None, [256], [0, 256]).flatten()  # Blue channel
    return hist_b / hist_b.sum()  # Normalize to unit sum

def predict_glucose_tflite(tflite_path, image_path, visualize=False):
    """
    Predict glucose level from a single image using the TFLite model.

    Args:
        tflite_path: Path to the TFLite model file.
        image_path: String, path to the input image.
        visualize: Boolean, whether to plot the histogram (default: False).

    Returns:
        Tuple: (image_path, prediction), prediction in mmol/L.
    """
    # Load TFLite model
    print("Loading TFLite model...")
    interpreter = tf.lite.Interpreter(model_path=tflite_path)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    # Load and preprocess image
    img = cv2.imread(image_path)
    if img is None:
        print(f"Warning: Image not found at {image_path}")
        return (image_path, None)

    # Resize to 640x480 (paper's resolution)
    img = cv2.resize(img, (640, 480))
    # Convert BGR to RGB
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Normalize to [0,1]
    img_normalized = img / 255.0
    # Compute blue channel histogram
    hist = compute_blue_histogram_features(img_normalized)
    # Reshape and convert to float32 for TFLite
    hist = hist.reshape(1, 256).astype(np.float32)

    # Predict glucose level
    interpreter.set_tensor(input_details[0]['index'], hist)
    interpreter.invoke()
    pred = interpreter.get_tensor(output_details[0]['index']).flatten()[0]

    # Visualize histogram if requested
    if visualize:
        plt.figure(figsize=(8, 4))
```

```
        plt.plot(hist.flatten(), color='blue')
        plt.title(f'Blue Channel Histogram ({image_path})\nPredicted Glucose: {pred:.2f} mmol/L')
        plt.xlabel('Bin')
        plt.ylabel('Normalized Intensity')
        plt.grid(True)
        plt.show()

    # Print prediction
    print(f"{image_path}: Predicted Glucose: {pred:.2f} mmol/L")

    return (image_path, pred)

# Example usage (uncomment and update paths as needed)
"""
tflite_path = "/path/to/model.tflite"
image_path = "/path/to/new_image.jpg"
prediction = predict_glucose_tflite(tflite_path, image_path, visualize=True)
"""
```

Out[10]: '\ntflite_path = "/path/to/model.tflite"\nimage_path = "/path/to/new_image.jpg"\nprediction = predict_glucose_tflite(tflite_path, image_path, visualize=True)\n'

In [11]:
```
tflite_path = "/kaggle/working/ann_model.tflite"
image_path = "/kaggle/input/4-4-sineth/Glyco_20250426_133243.jpg"
prediction = predict_glucose_tflite(tflite_path, image_path, visualize=True)
```

```
Loading TFLite model...
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```




```
/kaggle/input/4-4-sineth/Glyco_20250426_133243.jpg: Predicted Glucose: 5.41 mmol/L
```

In [ ]: