Final Project

MCS-DS, Name: Yibo Li, netID: yibol2, UIN: 675674020

One student (Yibo Li) worked on this assignment

Project Video: https://youtu.be/kPKwbpW8H0k

Github Repository: https://github.com/yiboli1990/CS-598-DL4H-Project

1 Introduction

1.1 Background of the Problem

- **Problem Type**: The focus is on predicting intraoperative hypotension (IOH), a significant perioperative challenge linked to increased risks such as acute kidney injury and myocardial injury. The predictive modeling applies to disease prediction, leveraging feature engineering and data processing of physiological signals.
- **Importance of Solving the Problem**: Accurate predictions of IOH can enable preemptive clinical interventions, enhancing patient safety and surgical outcomes.
- **Difficulty of the Problem**: The complexity in predicting IOH arises from the multifactorial nature of blood pressure influences during surgery, including patient-specific physiological responses and varying surgical and anesthetic factors. Traditional methods have often fallen short due to their inability to effectively integrate these dynamic factors.
- **State of the Art Methods**: Advances in machine learning have utilized more comprehensive physiological data analysis to enhance predictions. However, challenges remain in accurately modeling and predicting with these methods due to the complexity of the physiological data.

1.2 Paper Explanation

- **Proposed Method by the Paper**: The study presented a deep learning model that predicts IOH using arterial blood pressure (ABP), electrocardiogram (ECG), and electroencephalogram (EEG) waveforms. This approach facilitates a nuanced analysis of multiple physiological signals to anticipate hypotension.
- **Innovations of the Method**: The key innovation lies in combining multiple physiological waveforms and employing deep learning to predict IOH without the extensive feature engineering that traditional methods typically require.
- **Performance of the Proposed Method**: The model achieved high predictive performance, with area under the receiver operating characteristic curve (AUROC) values exceeding 0.97 (3 min before event), showcasing its potential utility in clinical settings.
- **Contribution to the Research Regime**: The study significantly contributes by enhancing the predictability of IOH through a novel data integration approach and advanced predictive modeling techniques, addressing gaps left by traditional methods and offering a robust and reliable prediction method.

2 Scope of Reproducibility

In this project, I aim to test the following hypotheses derived from the original paper and further detailed in my project proposal:

1. **Hypothesis 1**: Deep learning models are capable of accurately predicting instances of IOH by analyzing complex patterns embedded in physiological waveforms such as ABP, EEG, and ECG. This hypothesis is grounded in the premise that the nuanced dynamics of these signals, when combined, offer a comprehensive view of the patient's physiological state, enabling more precise predictions (Jo et al., 2022).

2. **Hypothesis 2**: The omission of ECG data, while maintaining ABP and EEG inputs, will not degrade and may even enhance the model's predictive accuracy for IOH. This hypothesis is based on findings from the original study, which indicated that models utilizing only ABP and EEG data performed comparably or slightly better than models integrating all three waveform types. This suggests that ABP and EEG alone may provide sufficient information for effective IOH prediction, potentially simplifying the data requirements without compromising the model’s performance.

2.1 Experiments to Test the Hypotheses

- **Experiment for Hypothesis 1**: I will develop and train a deep learning model using the specified physiological waveforms

Experiment for Hypothesis 1: I will develop and train a deep learning model using the specified physiological waveforms (ABP, EEG, ECG) to predict hypotensive events. The model's performance will be evaluated based on its ability to forecast these events accurately in a test dataset.

- **Experiment for Hypothesis 2**: I will conduct a comparative analysis by training two sets of models: one set using all three physiological waveforms (ABP, EEG, ECG) and another set using only ABP and EEG. The performance of these models will be rigorously evaluated on their ability to predict hypotensive events in a test dataset. The comparison will focus on key metrics such as accuracy, AUROC, and AUPRC to determine if models trained without ECG data indeed yield similar or improved predictive outcomes. This experiment aims to validate the original study's findings and explore the implications of data simplification in clinical predictive modeling.

These experiments are designed to validate the claims made in the original study and explored in my project proposal, thereby contributing to the understanding and development of robust predictive models for intraoperative hypotension.

3 Methodology

This section outlines the comprehensive methodology adopted to test the hypotheses stated earlier. It includes detailed descriptions and executable code necessary for replicating the experiments conducted in the original study and for any additional investigations outlined in the project proposal.

The methodology is structured into three main subsections:

- **Environment**: This subsection details the software environment used for all analyses and experiments in this project. It specifies the Python version and the necessary dependencies and packages required to replicate the experiments accurately.
- **Data**: This subsection will detail the processes involved in acquiring, preparing, and preprocessing the physiological data (ABP, EEG, and ECG). It covers everything from the initial data loading to the final steps required to prepare the data for modeling.
- **Model**: This subsection describes the construction, configuration, and training of the deep learning models used. It includes the architectural design of the models, the specifics of their implementation, and the techniques used for training and evaluating their performance on the task of predicting intraoperative hypotension (IOH).

Each subsection is designed to be self-contained, with annotations explaining the purpose and function of each code block, thereby ensuring clarity and reproducibility of the experiments.

3.1 Environment

To ensure the reproducibility of our experiments, the following software environment was used:

3.1.1 Python Version:

- **Python 3.8**: This version is chosen for its stability and wide support for data science libraries.

In [40]:
```
import sys
print("Python version")
print(sys.version)
print("Version info.")
print(sys.version_info)
```

```
Python version
3.8.19 (default, Mar 20 2024, 15:27:52)
[Clang 14.0.6 ]
Version info.
sys.version_info(major=3, minor=8, micro=19, releaselevel='final', serial=0)
```

3.1.2 Key Dependencies/Packages:

Below is a list of the primary Python libraries utilized in this project, which are essential for data manipulation, visualization, machine learning, and neural network modeling:

- **NumPy**: Provides support for efficient numerical computations.
- **Pandas**: Used for data manipulation and analysis.
- **Matplotlib**: For generating various plots and graphs.
- **Scikit-Learn**: Offers simple and efficient tools for predictive data analysis.
- **TensorFlow**: An end-to-end open-source platform for machine learning to build and train ML models.
- **VitalDB**: A toolkit for accessing and analyzing clinical data from the VitalDB dataset.
- **Requests**: Allows sending HTTP requests using Python, used here for downloading datasets.

Specific Packages and Their Roles:

Specific Packages and Their Roles:

- **vitaldb** for loading and handling clinical datasets.
- **numpy** and **pandas** for data manipulation and numerical operations.
- **matplotlib** and **sklearn.metrics** for visualization and metrics computation.
- **tensorflow** and its submodules (e.g., `keras.layers`, `keras.models`) for building and training deep learning models.
- **warnings** to manage warnings during code execution.

Note: Ensure that Python environment is set up with these packages to avoid any compatibility issues during code execution. The above packages are crucial for the procedures and results presented in this project.

Import Necessary Packages Analysis

In [41]:
```
import vitaldb
import numpy as np
import pandas as pd
import os
import random
import requests
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, roc_auc_score, average_precision_score
from sklearn.metrics import roc_curve, precision_recall_curve, auc
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, BatchNormalization, ReLU, Add
from tensorflow.keras.layers import MaxPooling1D, GlobalAveragePooling1D, Dense, Input
from tensorflow.keras.regularizers import l2
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.callbacks import ModelCheckpoint
import warnings
warnings.filterwarnings("ignore")
```

3.2 Data

This section describes the data acquisition, preprocessing, and preparation process used to analyze intraoperative hypotension (IOH) using physiological waveform data sourced from the VitalDB dataset.

Due to computational constraints, my data processing was limited to 200 cases, resulting in 133,592 segments. This subset ensures that the experiments are manageable while still providing significant insights.

The code presented in this section is a demonstration involving only 5 cases. This is for illustrative purposes to show the process without the extensive runtime required for the full dataset. The actual processed data consists of the 200 cases mentioned earlier and is stored in `preprocessed_data_200.npz`. You can download the dataset (200 cases) from my Google Drive using this link https://drive.google.com/file/d/11uUXAva6AEhwuHrwi4iqhAZ6WiKvVIs_/view?usp=sharing

Due to size of the data (10GB processed data), it has not been uploaded to the Github or other cloud drive

3.2.1 Data Acquisition and Description

The data for this study consists of physiological waveforms, including arterial blood pressure (ABP), electroencephalogram (EEG), and electrocardiogram (ECG), collected from surgical patients. While the original paper by Jo et al. claims to have used data from 39,600 cases, the publicly available dataset from VitalDB that I accessed contains just 6,388 cases.

The VitalDB dataset is a rich resource for clinical data collected during anesthesia, making it highly relevant for studying intraoperative conditions such as hypotension.

In [42]:
```
# Load the track list and case information from VitalDB
df_trks = pd.read_csv('https://api.vitaldb.net/trks')
df_cases = pd.read_csv("https://api.vitaldb.net/cases")
```

3.2.2 Data Visulalization

The VitalDB dataset is a comprehensive resource utilized for analyzing various intraoperative conditions, notably intraoperative hypotension. It includes two principal components:

- **Track List (`df_trks`)**: This dataframe contains metadata for different physiological signals recorded during surgeries. Each record is identified by parameters such as `caseid` (a unique identifier for each surgical case), `tname` (track name indicating the type of physiological data), and additional attributes relevant to the recording context.
- **Case Information (`df_cases`)**: This dataframe details each surgical case, providing information on `caseid`, `age`, `sex`

- **Case Information (`df_cases`)**: This dataframe details each surgical case, providing information on `caseid`, `age`, `sex`, `weight`, `height`, `ane_type` (type of anesthesia used), and `optype` (type of operation), among others. Such data is crucial for filtering suitable cases for analysis and understanding the demographic and clinical context of the recorded data.

Visualizations

1. Age Distribution of Patients

- **Description**: The histogram below illustrates the age distribution of patients within the dataset. This visualization helps in understanding the age diversity of the patient population involved in the surgical cases.
- **Insights**:
  - The age distribution primarily spans from 20 to 80 years, with a noticeable concentration in middle-aged adults.
  - This wide range in age helps ensure that the study's findings are applicable across a diverse adult population.

2. Distribution of Operation Types

- **Description**: The bar chart provided next displays the count of various types of operations performed, as recorded in the dataset. Each bar corresponds to a specific type of surgery, highlighting its frequency within the collected data.
- **Insights**:
  - Common surgeries like 'General' and 'Orthopedic' are predominant, reflecting typical clinical scenarios where intraoperative monitoring is crucial.
  - The diversity in surgery types underscores the dataset's utility in developing models that are adaptable to different surgical contexts, particularly those with higher risks of complications such as hypotension.

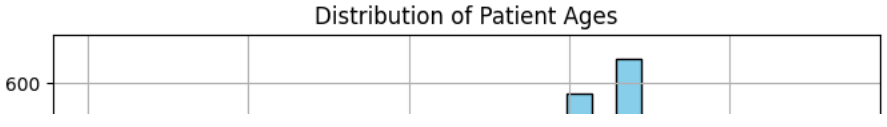
3. Anesthesia Types Used

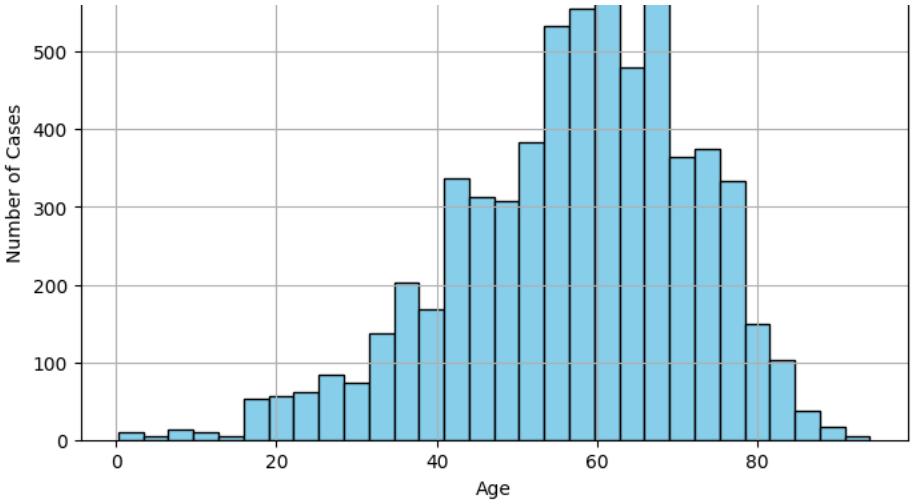
- **Description**: The pie chart represents the distribution of different types of anesthesia used during the surgical procedures. This provides insights into the anesthesia practices and their prevalence within the dataset.
- **Insights**:
  - The majority of cases utilize general anesthesia, indicating its common application in surgeries requiring intensive monitoring and control over physiological parameters.
  - The presence of various anesthesia types allows for analyzing the impact of anesthesia techniques on intraoperative hypotension, enriching the predictive model's applicability and accuracy.

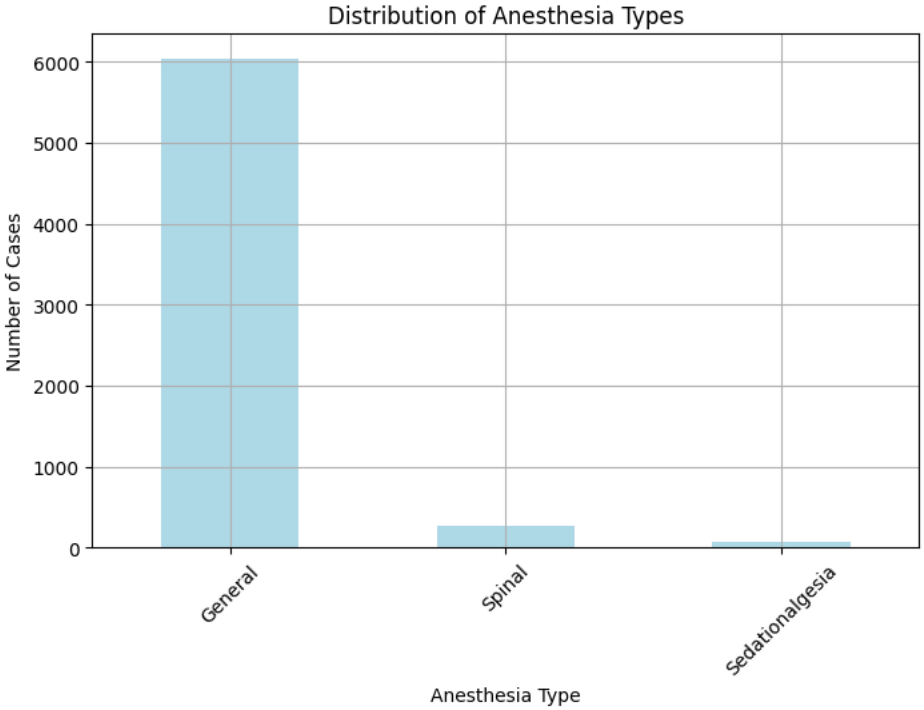
These charts are instrumental for evaluating the dataset's adequacy and scope concerning the study of intraoperative conditions, ensuring that the predictive models developed are both robust and generalizable to real-world clinical settings.

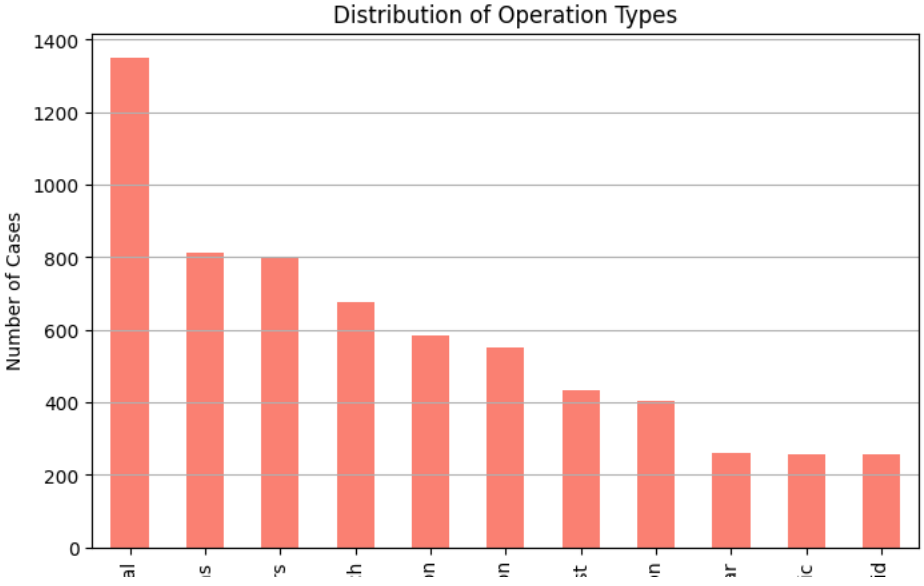
In [43]:

```
# Histogram of ages in the cases data
plt.figure(figsize=(8, 5))
plt.hist(df_cases['age'].dropna(), bins=30, color='skyblue', edgecolor='black')
plt.title('Distribution of Patient Ages')
plt.xlabel('Age')
plt.ylabel('Number of Cases')
plt.grid(True)
plt.show()

# Bar chart of anesthesia types
anesthesia_counts = df_cases['ane_type'].value_counts()
plt.figure(figsize=(8, 5))
anesthesia_counts.plot(kind='bar', color='lightblue')
plt.title('Distribution of Anesthesia Types')
plt.xlabel('Anesthesia Type')
plt.ylabel('Number of Cases')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# Count of each operation type
operation_counts = df_cases['optype'].value_counts()
plt.figure(figsize=(8, 5))
operation_counts.plot(kind='bar', color='salmon')
plt.title('Distribution of Operation Types')
plt.xlabel('Operation Type')
plt.ylabel('Number of Cases')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.show()
```




Number of Cases
Age
0
20
40
60
80
100
200
300
400
500


Distribution of Anesthesia Types
Number of Cases
Anesthesia Type
General
Spinal
Sedationalgesia
0
1000
2000
3000
4000
5000
6000


Distribution of Operation Types
Number of Cases
0
200
400
600
800
1000
1200
1400

3.2.3 Data Filtering and Preprocessing

The preprocessing steps include selecting cases based on specific criteria, checking the availability of required waveforms, and ensuring signal quality. Here, I focu on adult cases with general anesthesia, excluding any cardiac surgery types to standardize the dataset and improve the relevance of the predictions for non-cardiac procedures, which is the same approach used by the original research.

Filtering Criteria

- **Age**: 18 years and older.
- **Anesthesia Type**: General.
- **Operation Type**: Non-cardiac.

Waveform Presence

To ensure the integrity of our analysis, we include only those cases that have the necessary waveforms (ABP, ECG, and EEG).

In [44]:
```
# Functions for waveform presence
def has_required_waveforms(case_id, required_waveforms):
    case_waveforms = set(df_trks[df_trks['caseid'] == case_id]['tname'])
    return all(waveform in case_waveforms for waveform in required_waveforms)

# Define required waveforms for inclusion
required_waveforms = ['SNUADC/ART', 'SNUADC/ECG_II', 'BIS/EEG1_WAV']

# Apply inclusion and exclusion criteria
caseids = set(df_cases[(df_cases['age'] >= 18) &
                       (df_cases['ane_type'] == 'General') &
                       (df_cases['optype'] != 'Cardiac')]['caseid'])

# Filter caseids to include only those with the required waveforms
caseids = [case_id for case_id in caseids if has_required_waveforms(case_id, required_waveforms)]

random.seed(4020)
print(f'Total {len(caseids)} cases found')
np.random.shuffle(caseids)  # Shuffle caseids
```

```
Total 3307 cases found
```

3.2.4 Data Segmentation (5 cases for demonstration)

Data segmentation involves breaking down continuous waveform data into manageable, discrete segments for analysis. Each segment is a one-minute window of the waveform data, designed to predict hypotension three minutes ahead of its occurrence.

In this section, we will read the MBP data while iterating caseids. The arterial waveform data of each case will be read by passing the caseid into the vitaldb python library. Then, the pairs of x (input) and y (label) for training and validation of the model will be extracted.

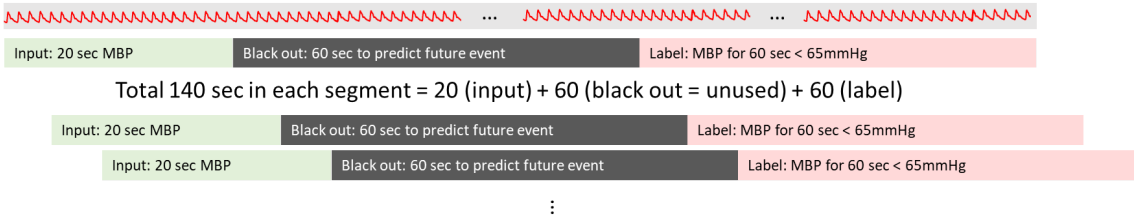
Note: Data Processing Simplifications

In adapting the methodology from the original study to suit our project's constraints, we implemented several key simplifications:

- **Waveform Image Quality**: We did not evaluate the quality of waveform images due to our utilization of a smaller dataset, diverging from the original study's approach where this was a standard procedure.
- **Sampling Rate**: Computational limitations necessitated a reduction in the sampling rate from the original 500 Hz to 25 Hz. This change helps manage computational demands while maintaining essential data integrity.
- **Analysis Window**: Our analysis was specifically focused on the 3 minutes immediately preceding each hypotensive event, rather than a broader time frame. This focus was chosen to capture the most critical dynamics leading up to hypotensive episodes, optimizing our data processing efforts.

These modifications were essential to ensure that our project was feasible within the constraints of available resources, while still striving to closely approximate the critical analyses of the original study.

Below chart is for illustration purposes only:



In [45]:

```
MINUTES_AHEAD = 3  # Predict hypotension 3 minute ahead
SRATE = 25  # Sampling rate for the signals
WINDOW_SIZE = 60  # seconds in each input segment (1 minute window to match event definition)
SEGMENT_STEP = 10 * SRATE  # 10 seconds step between segments
MAX_CASES = 5  # For illustrative purposes. Real dataset has 200 cases

x = []  # input with shape of (segments, timepoints, channels)
y = []  # output with shape of (segments)
valid_mask = []  # validity of each segment
c = []  # caseid of each segment

random.seed(4020)
caseids = np.random.choice(caseids, MAX_CASES, replace=False)  # Randomly select MAX_CASES without replacement
total_cases = len(caseids)

# Load cases up to MAX_CASES
for index, caseid in enumerate(caseids):
    print(f'Loading case {index + 1} of {total_cases} (ID: {caseid})...', flush=True)

    # Read the waveforms
    arts = vitaldb.load_case(caseid, ['SNUADC/ART'], 1 / SRATE).flatten()
    ecgs = vitaldb.load_case(caseid, ['SNUADC/ECG_II'], 1 / SRATE).flatten()
    eegs = vitaldb.load_case(caseid, ['BIS/EEG1_WAV'], 1 / SRATE).flatten()

    # Iterate over each segment
    for i in range(0, len(arts) - SRATE * (WINDOW_SIZE + (1 + MINUTES_AHEAD) * 60), SEGMENT_STEP):
        segx_arts = arts[i:i + SRATE * WINDOW_SIZE]
        segx_ecgs = ecgs[i:i + SRATE * WINDOW_SIZE]
        segx_eegs = eegs[i:i + SRATE * WINDOW_SIZE]

        # Hypotension label calculation
        end_idx = i + SRATE * (WINDOW_SIZE + MINUTES_AHEAD * 60)
        label_idx = end_idx + SRATE * 60
        if label_idx > len(arts):
            continue
        segy = arts[end_idx:label_idx]

        # 2-sec moving average for arterial waveform to decide hypotension event
        n = 2 * SRATE
        segy_ma = np.convolve(segy, np.ones(n)/n, mode='valid')
        evt = np.nanmin(segy_ma) < 65  # True if hypotension

        # Combine the three channels into one array
        segx_combined = np.stack([segx_arts, segx_ecgs, segx_eegs], axis=-1)

        # Basic validity check
        valid = np.all([
            np.nanmean(segx_arts) <= 200, np.nanmean(segx_arts) >= 30,
            np.nanmax(segx_arts) - np.nanmin(segx_arts) > 30,
            not np.any(np.abs(np.diff(segx_arts)) > 30)
        ])

        x.append(segx_combined)
        y.append(evt)
        valid_mask.append(valid)
        c.append(caseid)

    print(f"{len(x)} segments collected so far")

# Convert lists to numpy arrays
x = np.array(x)
y = np.array(y).astype(int)
valid_mask = np.array(valid_mask)
c = np.array(c)

# Handle missing values by forward filling, then backfilling
x = np.nan_to_num(x, nan=np.nanmean(x))  # Replace NaNs with channel mean
```

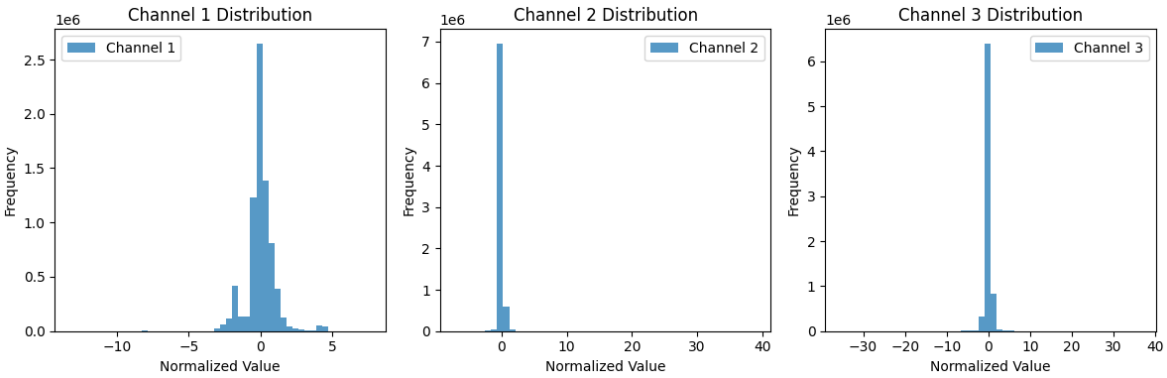
```
# Normalize x: zero mean and unit variance
x -= np.mean(x, axis=0)
x /= np.std(x, axis=0)

# Save the data for later use
np.savez_compressed('preprocessed_data_5.npz', x=x, y=y, valid_mask=valid_mask, case_ids=c)
print("Data saved. Shapes:", x.shape, y.shape)
```

```
Loading case 1 of 5 (ID: 5696)...
753 segments collected so far
Loading case 2 of 5 (ID: 2935)...
2554 segments collected so far
Loading case 3 of 5 (ID: 1001)...
3459 segments collected so far
Loading case 4 of 5 (ID: 6089)...
4434 segments collected so far
Loading case 5 of 5 (ID: 781)...
5114 segments collected so far
Data saved. Shapes: (5114, 1500, 3) (5114,)
```

3.2.5 Plotting figures for illustration

In [47]:

```
plt.figure(figsize=(12, 4))
for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.hist(x[:, :, i].flatten(), bins=50, alpha=0.75, label=f'Channel {i+1}')
    plt.title(f'Channel {i+1} Distribution')
    plt.xlabel('Normalized Value')
    plt.ylabel('Frequency')
    plt.legend()
plt.tight_layout()
plt.show()
```



3.3 Model

3.3.1 Citation to the Original Paper

This project builds upon the methodologies described in the study "Predicting intraoperative hypotension using deep learning with waveforms of arterial blood pressure, electroencephalogram, and electrocardiogram: Retrospective study" by Yong-Yeon Jo, Jong-Hwan Jang, Joon-myoung Kwon, and colleagues. The study was published in PLoS ONE in August 2022. The authors developed deep learning models to predict intraoperative hypotension using physiological waveforms, providing a foundation for further exploration and development in this field. The study can be accessed via the DOI: 10.1371/journal.pone.0272055.

- **Reference**: Jo, Y.-Y., Jang, J.-H., Kwon, J.-m., et al. (2022). Predicting intraoperative hypotension using deep learning with waveforms of arterial blood pressure, electroencephalogram, and electrocardiogram: Retrospective study. *PLoS ONE*, 17(8): e0272055. doi:10.1371/journal.pone.0272055.

3.3.2 Link to the Original Paper's Repository

No repository link is available for the original paper's implementation. This section of the report is provided to acknowledge the sources used, and since the original implementation details are not publicly shared through any repository, replication and comparisons were based solely on the descriptions and methods outlined in the published paper.

3.3.3 Model Descriptions

This section outlines the architecture and training process of the deep learning model used in this project to predict intraoperative hypotension (IOH). The model is a Residual Network (ResNet), which is particularly suitable for processing time-series data like the physiological waveforms used in this study.

3.3.3.1 Model Architecture

The model utilizes residual blocks, which help in mitigating the vanishing gradient problem by allowing gradients to flow through the network via skip connections. Each block consists of convolutional layers, batch normalization, and ReLU activations.

3.3.3.1.1 Residual Block Description

- **Convolutional Layers**: These layers extract features by applying filters to the input. The use of 1D convolution is appropriate for time-series data.
- **Batch Normalization**: This layer normalizes the activations from the previous layer, which helps in speeding up the training process and reducing the sensitivity to network initialization.
- **ReLU Activation**: This activation function introduces non-linearity into the model, allowing it to learn more complex patterns.
- **Shortcut Connections**: These are used to add the input of the residual block back to its output, which helps in training deeper networks effectively.

3.3.3.1.2 Model Summary

- **Layer Sizes and Types**: The model starts with a 64-filter convolutional layer followed by a max pooling layer to reduce dimensionality. This is followed by another residual block with 128 filters.
- **Activation Function**: The output layer uses a sigmoid activation function, suitable for binary classification tasks.
- **Regularization**: L2 regularization is applied to the convolutional layers to prevent overfitting.

3.3.3.2 Training Objectives

- **Loss Function**: The model uses binary cross-entropy as the loss function, which is standard for binary classification tasks.
- **Optimizer**: The Adam optimizer is used for training, known for its effectiveness and efficiency in handling sparse gradients and adapting the learning rate.
- **Metrics**: Accuracy is tracked as a performance metric to evaluate the model during training and validation.

3.3.3.3 Additional Configurations

- **Pretrained Models**: For this project, the model is trained from scratch. Pretrained models could be incorporated in future to compare performance or improve training efficiency.
- **Uncertainty Analysis**: While not implemented here, Monte Carlo simulations could be used in future work to assess the uncertainty in the model's predictions, providing insights into the confidence level of the predictions.

3.3.4 Model Implementation Code

In [48]:

```
def residual_block(x, filters, kernel_size, strides=1):
    shortcut = x
    if strides > 1 or x.shape[-1] != filters:
        shortcut = Conv1D(filters, 1, strides=strides, padding="same", kernel_regularizer=l2(1e-3))(shortcut)

    # Main path
    x = Conv1D(filters, kernel_size, strides=strides, padding="same", kernel_regularizer=l2(1e-3))(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv1D(filters, kernel_size, padding="same", kernel_regularizer=l2(1e-3))(x)
    x = BatchNormalization()(x)

    # Add shortcut
    x = Add()([x, shortcut])
    x = ReLU()(x)
    return x

def build_resnet(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = residual_block(inputs, 64, 8, strides=1)  # Start with stride 1 to maintain dimension
    x = MaxPooling1D(pool_size=2)(x)
    x = residual_block(x, 128, 8, strides=2)  # Use stride 2 here to reduce dimension
    x = GlobalAveragePooling1D()(x)
    outputs = Dense(num_classes, activation='sigmoid')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model

# Build the model
model = build_resnet((1500, 3), 1)
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "model_4"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_5 (InputLayer)        [(None, 1500, 3)]            0         []

 conv1d_20 (Conv1D)          (None, 1500, 64)             1600      ['input_5[0][0]']

 batch_normalization_13 (Ba  (None, 1500, 64)             256       ['conv1d_20[0][0]']
 tchNormalization)

 re_lu_13 (ReLU)             (None, 1500, 64)             0         ['batch_normalization_13[0][0]
                                                                    ']

 conv1d_21 (Conv1D)          (None, 1500, 64)             32832     ['re_lu_13[0][0]']

 batch_normalization_14 (Ba  (None, 1500, 64)             256       ['conv1d_21[0][0]']
 tchNormalization)

 conv1d_19 (Conv1D)          (None, 1500, 64)             256       ['input_5[0][0]']

 add_6 (Add)                 (None, 1500, 64)             0         ['batch_normalization_14[0][0]
                                                                    ',
                                                                     'conv1d_19[0][0]']

 re_lu_14 (ReLU)             (None, 1500, 64)             0         ['add_6[0][0]']

 max_pooling1d_4 (MaxPoolin  (None, 750, 64)              0         ['re_lu_14[0][0]']
 g1D)

 conv1d_23 (Conv1D)          (None, 375, 128)             65664     ['max_pooling1d_4[0][0]']

 batch_normalization_15 (Ba  (None, 375, 128)             512       ['conv1d_23[0][0]']
 tchNormalization)

 re_lu_15 (ReLU)             (None, 375, 128)             0         ['batch_normalization_15[0][0]
                                                                    ']

 conv1d_24 (Conv1D)          (None, 375, 128)             131200    ['re_lu_15[0][0]']

 batch_normalization_16 (Ba  (None, 375, 128)             512       ['conv1d_24[0][0]']
 tchNormalization)

 conv1d_22 (Conv1D)          (None, 375, 128)             8320      ['max_pooling1d_4[0][0]']

 add_7 (Add)                 (None, 375, 128)             0         ['batch_normalization_16[0][0]
                                                                    ',
                                                                     'conv1d_22[0][0]']

 re_lu_16 (ReLU)             (None, 375, 128)             0         ['add_7[0][0]']

 global_average_pooling1d_4  (None, 128)                  0         ['re_lu_16[0][0]']
  (GlobalAveragePooling1D)

 dense_4 (Dense)             (None, 1)                    129       ['global_average_pooling1d_4[0
                                                                    ][0]']

==================================================================================================
Total params: 241537 (943.50 KB)
Trainable params: 240769 (940.50 KB)
Non-trainable params: 768 (3.00 KB)
__________________________________________________________________________________________________
```

3.3.5 Pretrained Model

The deep learning model developed for predicting intraoperative hypotension was pretrained and optimized prior to evaluation. The saved model checkpoint can be accessed through the project's GitHub repository, enabling replication and further exploration of the model's capabilities. The checkpoint contains the model weights that were found to be most effective during the validation phase.

Checkpoint Link: Pretrained Model Checkpoint

This resource allows anyone to load the pretrained model without the need to retrain from scratch, facilitating quick deployment and testing of the model's performance on similar datasets or under different experimental conditions.

3.4 Training (Code in this section has been commented out; (runtime:

~3hr))

This section discusses the model training process including data preparation, computational requirements, and the training execution itself.

3.4.1 Hyperparameters

Hyperparameters play a crucial role in defining the training dynamics and the effectiveness of the model. The following are key hyperparameters used during the model training:

- **Batch Size**: 32, balancing the computational efficiency and model convergence.
- **Learning Rate**: An initial value of 0.001 was used, with adjustments made by the optimizer over training epochs.
- **Optimizer**: Adam optimizer was chosen for its efficiency in handling sparse gradients and adaptive learning rate capabilities.
- **Epochs**: 10, ensuring sufficient exposure of the model to the training data while preventing overfitting.
- **Loss Function**: Binary cross-entropy, suitable for binary classification tasks.
- **Dropout Rate**: Applied at 0.5 in applicable layers to reduce overfitting.
- **Activation Function**: ReLU in hidden layers for non-linear transformations, with a Sigmoid output layer for binary classification.

3.4.2 Computational Requirements

The computational framework and resources used for training are as follows:

- **Computer**: Apple M1 MacBook Pro
- **CPU**: Apple M1 Pro
- **GPU**: Apple M1 Pro Built-in
- **RAM**: 16 GB
- **Storage**: SSD with at least 50GB free space for efficient data handling and model training
- **Runtime**: **Around 3 hours**
- **Average Runtime per Epoch**: Approximately 18 minutes per epoch, reflecting the computational load and complexity of the training process.
- **GPU Hours Used**: Considering the total runtime and the involvement of GPU resources, the total GPU hours used was approximately 3 hours for the full training process.
- **Training Epochs**: 10 epochs, consistent with the need to balance model accuracy and overfitting.

3.4.3 Training Code

3.4.3.1 Data Split

Before training, the data is split into training, validation, and testing sets to evaluate the model's performance effectively and avoid overfitting. The data is split based on a 6:1:3 ratio among training, validation, and test sets respectively, ensuring that each split is done on a per-case basis to prevent data leakage.

In [11]:
```
"""
# Load data
data = np.load('preprocessed_data_200.npz')
x = data['x']
y = data['y']
valid_mask = data['valid_mask']
c = data['case_ids']

# Unique case identifiers and shuffle
unique_cases = np.unique(c)
np.random.shuffle(unique_cases)

# Calculate split indices, ensuring at least 1 case for validation and test when possible
n_cases = len(unique_cases)
n_train = int(n_cases * 0.6)
n_val = max(int(n_cases * 0.1), 1)  # Ensure at least 1 case for validation if possible
n_test = max(n_cases - n_train - n_val, 1)  # Ensure at least 1 case for test

train_cases = unique_cases[:n_train]
val_cases = unique_cases[n_train:n_train + n_val]
test_cases = unique_cases[n_train + n_val:]

# Function to extract data by cases
def extract_data(cases):
    indices = np.isin(c, cases)
    return x[indices], y[indices]

x_train, y_train = extract_data(train_cases)
x_val, y_val = extract_data(val_cases)
```

```
x_test, y_test = extract_data(test_cases)

# Save x_test and y_test as CSV files
np.savetxt("x_test.csv", x_test.reshape(x_test.shape[0], -1), delimiter=",")
np.savetxt("y_test.csv", y_test, delimiter=",")

print(f'Training data: {x_train.shape}, Validation data: {x_val.shape}, Test data: {x_test.shape}')
"""
```

```
Training data: (164808, 1500, 3), Validation data: (27890, 1500, 3), Test data: (82780, 1500, 3)
```

In [12]:

```
"""
print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"x_val shape: {x_val.shape}")
print(f"y_val shape: {y_val.shape}")
"""
```

```
x_train shape: (164808, 1500, 3)
y_train shape: (164808,)
x_val shape: (27890, 1500, 3)
y_val shape: (27890,)
```

3.4.3.2 Model Training

With the data prepared, the next step involves training the model using the specified batch size and epochs. Monitoring both training and validation performance helps in tuning and early stopping if needed.

In [13]:

```
"""
# Define the path where the checkpoints will be saved
checkpoint_filepath = 'model_checkpoint.h5'

# Create the ModelCheckpoint callback
checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,  # Save the entire model
    monitor='val_accuracy',    # Monitor the validation accuracy
    save_best_only=True,       # Save only the best model
    verbose=1                  # Log the progress
)
"""
```

In [14]:

```
"""
# Define training parameters
BATCH_SIZE = 32
EPOCHS = 10

# Training the model with the checkpoint callback
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=[checkpoint_callback],  # Include the callback here
    verbose=1
)

# Plot training history
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Training History')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
"""
```

```
Epoch 1/10
2024-05-04 21:22:35.347544: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
5151/5151 [==============================] - ETA: 0s - loss: 0.4795 - accuracy: 0.8107
2024-05-04 21:26:45.629253: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
Epoch 1: val_accuracy improved from -inf to 0.83525, saving model to model_checkpoint.h5
5151/5151 [==============================] - 277s 53ms/step - loss: 0.4795 - accuracy: 0.8107 - val_loss: 0.399
9 - val_accuracy: 0.8352
Epoch 2/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4408 - accuracy: 0.8161
Epoch 2: val_accuracy improved from 0.83525 to 0.86167, saving model to model_checkpoint.h5
```

```
5151/5151 [==============================] - 280s 54ms/step - loss: 0.4408 - accuracy: 0.8161 - val_loss: 0.366
2 - val_accuracy: 0.8617
Epoch 3/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4358 - accuracy: 0.8172
Epoch 3: val_accuracy did not improve from 0.86167
5151/5151 [==============================] - 270s 52ms/step - loss: 0.4358 - accuracy: 0.8172 - val_loss: 0.406
5 - val_accuracy: 0.8180
Epoch 4/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4304 - accuracy: 0.8185
Epoch 4: val_accuracy did not improve from 0.86167
5151/5151 [==============================] - 266s 52ms/step - loss: 0.4304 - accuracy: 0.8185 - val_loss: 0.357
2 - val_accuracy: 0.8615
Epoch 5/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4272 - accuracy: 0.8191
Epoch 5: val_accuracy improved from 0.86167 to 0.86296, saving model to model_checkpoint.h5
5151/5151 [==============================] - 271s 53ms/step - loss: 0.4272 - accuracy: 0.8191 - val_loss: 0.354
5 - val_accuracy: 0.8630
Epoch 6/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4249 - accuracy: 0.8198
Epoch 6: val_accuracy did not improve from 0.86296
5151/5151 [==============================] - 264s 51ms/step - loss: 0.4249 - accuracy: 0.8198 - val_loss: 0.353
0 - val_accuracy: 0.8617
Epoch 7/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4230 - accuracy: 0.8215
Epoch 7: val_accuracy did not improve from 0.86296
5151/5151 [==============================] - 268s 52ms/step - loss: 0.4230 - accuracy: 0.8215 - val_loss: 0.353
1 - val_accuracy: 0.8621
Epoch 8/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4201 - accuracy: 0.8221
Epoch 8: val_accuracy improved from 0.86296 to 0.86371, saving model to model_checkpoint.h5
5151/5151 [==============================] - 274s 53ms/step - loss: 0.4201 - accuracy: 0.8221 - val_loss: 0.349
7 - val_accuracy: 0.8637
Epoch 9/10
5151/5151 [==============================] - ETA: 0s - loss: 0.4180 - accuracy: 0.8226
Epoch 9: val_accuracy did not improve from 0.86371
5151/5151 [==============================] - 271s 53ms/step - loss: 0.4180 - accuracy: 0.8226 - val_loss: 0.353
1 - val_accuracy: 0.8588
Epoch 10/10
5150/5151 [==============================>.] - ETA: 0s - loss: 0.4165 - accuracy: 0.8239
Epoch 10: val_accuracy did not improve from 0.86371
5151/5151 [==============================] - 274s 53ms/step - loss: 0.4165 - accuracy: 0.8239 - val_loss: 0.350
7 - val_accuracy: 0.8616
```

4 Evaluation

Note 1: Check point loaded from Github Repo

Note 2: Use the following commands to download the test datasets before running (full instruction on how to run the code has been included in the README file):

```
curl -L 'https://drive.google.com/uc?export=download&id=11ECqyTTMq8aGqw0g9RQFH6Fm-Mrn7TYm' -o
x_test.csv
```

```
curl -L 'https://drive.google.com/uc?export=download&id=18gxmUysPZKJspbLDRK5H1ZjmYUuQYe8D' -o
y_test.csv
curl -L 'https://drive.google.com/uc?export=download&id=1JbT4kyRl8irsf__yDPT6XfevD67OAKnd' -o
x_test_abp_eeg.csv
curl -L 'https://drive.google.com/uc?export=download&id=1G5iioEM8nDnxLTmpRBwodgoP4-ANoUl1' -o
y_test_abp_eeg.csv
```

In this section, we present the results obtained from the evaluation of the model trained to predict intraoperative hypotension (IOH) using physiological waveforms. The metrics reported here include accuracy, area under the receiver operating characteristic curve (AUROC), area under the precision-recall curve (AUPRC), as well as sensitivity and specificity. Additionally, graphical representations of the ROC and Precision-Recall curves are provided to visually assess model performance.

4.1 Metrics Descriptions

- **Accuracy**: This measures the proportion of true results (both true positives and true negatives) among the total number of cases examined.
- **AUROC (Area Under the Receiver Operating Characteristic Curve)**: This metric evaluates the model's ability to distinguish between classes at various threshold settings. AUROC is a value between 0 and 1, where 1 indicates perfect discrimination and 0.5 suggests no discrimination.
- **AUPRC (Area Under the Precision-Recall Curve)**: This reflects the trade-off between precision and recall for different threshold values. It is particularly useful when classes are imbalanced.
- **Sensitivity**: Also known as the true positive rate, measures the proportion of actual positives that are correctly identified as such.
- **Specificity**: Also known as the true negative rate, measures the proportion of actual negatives that are correctly identified as such.
- **Optimal Threshold**: This is the threshold value that maximizes the difference between the true positive rate and the false positive rate (J-statistic), enhancing the model's decision-making precision.

4.2 Evaluation Code

The following code demonstrates the model evaluation process, including the loading of test data, model weights, and the computation of metrics.

4.2.1 Test Accuracy

In [49]:

```
warnings.filterwarnings("ignore")
# URL of the model file on GitHub
url = 'https://github.com/yiboli1990/CS-598-DL4H-Project/raw/main/model_checkpoint.h5'

# Load x_test and y_test from CSV files
x_test = np.loadtxt("x_test.csv", delimiter=",")
y_test = np.loadtxt("y_test.csv", delimiter=",")

# Reshape x_test back to its original shape if necessary
x_test = x_test.reshape(-1, 1500, 3)  # Update dimensions according to your original data shape

# Make a GET request to fetch the raw content by the URL
r = requests.get(url)
with open('downloaded_model.h5', 'wb') as f:
    f.write(r.content)

print("Model downloaded successfully.")

# Load the downloaded model
best_model = load_model('downloaded_model.h5')

# Evaluate the model on the test data
test_loss, test_acc = best_model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc * 100:.2f}%")
```

```
Model downloaded successfully.
2024-05-07 21:30:30.760445: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
2587/2587 [==============================] - 90s 34ms/step - loss: 0.4253 - accuracy: 0.8206
Test Accuracy: 82.06%
```

4.2.2 Probability Predictions and Threshold Analysis

I then proceed to predict probabilities for the test data and analyze the model's performance across different thresholds:

In [50]:

```
# Predict using the best model loaded from the checkpoint
y_prob = best_model.predict(x_test)  # Get model predictions as probabilities
auroc = roc_auc_score(y_test, y_prob)
auprc = average_precision_score(y_test, y_prob)

def calculate_sensitivity_specificity(y_true, y_scores, threshold):
    y_pred = (y_scores >= threshold).astype(int)
    cm = confusion_matrix(y_true, y_pred)
    sensitivity = cm[1, 1] / (cm[1, 0] + cm[1, 1])  # TP / (FN + TP)
    specificity = cm[0, 0] / (cm[0, 1] + cm[0, 0])  # TN / (FP + TN)
    return sensitivity, specificity

fpr, tpr, roc_thresholds = roc_curve(y_test, y_prob)
# Find the optimal threshold using roc_thresholds
J = tpr - fpr
optimal_idx = np.argmax(J)
optimal_threshold = roc_thresholds[optimal_idx]
sensitivity, specificity = calculate_sensitivity_specificity(y_test, y_prob, optimal_threshold)

# Create a formatted string output
results_summary = f"""
Results Summary:
- AUROC (Area Under the ROC Curve): {auroc:.4f}
- AUPRC (Area Under the Precision-Recall Curve): {auprc:.4f}
- Sensitivity: {sensitivity:.4f}
- Specificity: {specificity:.4f}
- Optimal threshold based on J-statistic: {optimal_threshold:.4f}
"""

print(results_summary)
```

```
2024-05-07 21:32:25.617031: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
2587/2587 [==============================] - 32s 12ms/step

Results Summary:
- AUROC (Area Under the ROC Curve): 0.8679
- AUPRC (Area Under the Precision-Recall Curve): 0.7759
- Sensitivity: 0.7507
- Specificity: 0.8427
- Optimal threshold based on J-statistic: 0.2311
```

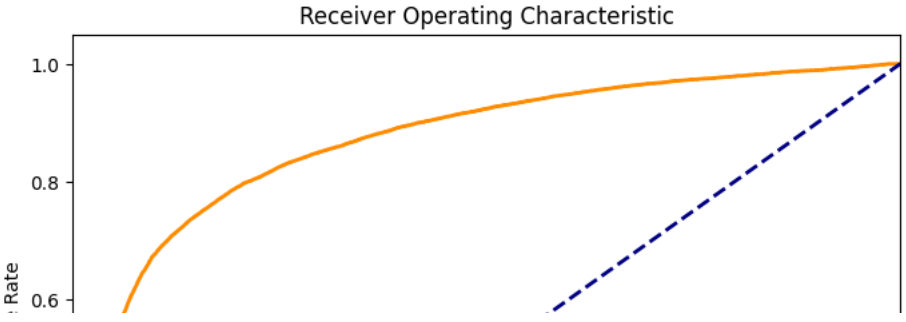
4.2.3 Visualization of Model Performance

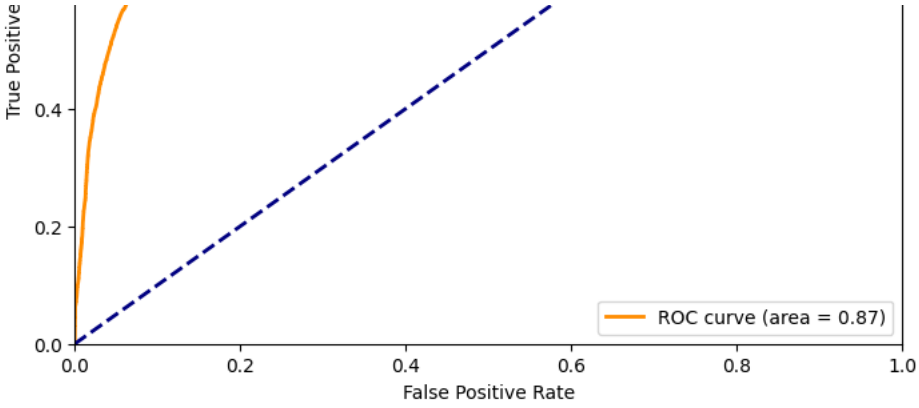
Visual representations of the ROC and Precision-Recall curves offer an intuitive understanding of performance across various thresholds:

ROC Curve

In [51]:

```
# Calculate ROC curve

roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

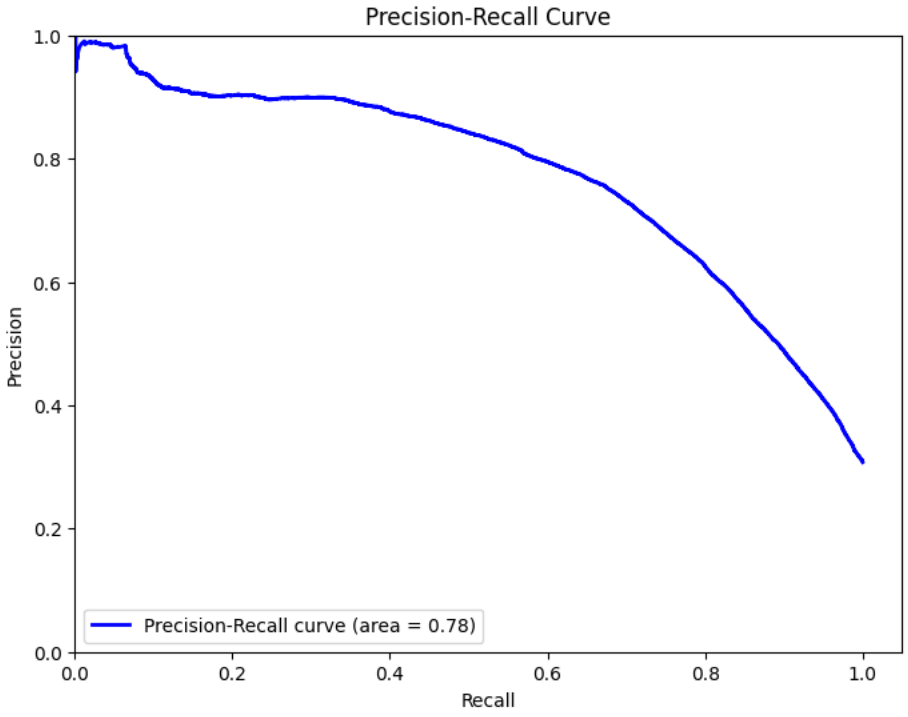




Precision-Recall Curve

In [52]:

```
# Calculate Precision-Recall curve
precision, recall, pr_thresholds = precision_recall_curve(y_test, y_prob)
pr_auc = auc(recall, precision)

# Plot Precision-Recall Curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve (area = %0.2f)' % pr_auc)
plt.xlim([0.0, 1.05])
plt.ylim([0.0, 1.0])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()
```



5 Results

5.1 Table of Results

This section presents a summary of the evaluation metrics obtained after testing the predictive model. The results help to determine how well the model can predict intraoperative hypotension using the selected physiological signals. The summary table provides a clear overview of the model's performance across different statistical metrics.

provides a clear overview of the model's performance across different statistical metrics.

Metric	Value
Test Accuracy	82.06%
AUROC (Area Under the ROC Curve)	0.8679
AUPRC (Area Under the Precision-Recall Curve)	0.7759
Sensitivity	0.7507
Specificity	0.8427
Optimal Threshold	0.2311

Summary

The model demonstrates a solid performance with an AUROC of 0.8679, indicating a high capability to distinguish between classes effectively. The AUPRC score of 0.7759 is satisfactory, reflecting the model's precision and recall balance at various threshold levels. Sensitivity at 0.7507 shows that the model is reasonably good at detecting true positives, which is critical for clinical applications where missing true cases of hypotension could be detrimental. The specificity of 0.8427 assures that the model effectively identifies negatives, thus minimizing false alarms. The optimal threshold derived from the J-statistic is 0.2311, optimizing the trade-off between sensitivity and specificity for practical use.

5.2 Conclusion for Hypothesis 1

The results from the evaluation of the deep learning model strongly support Hypothesis 1, which posited that deep learning models are capable of accurately predicting instances of intraoperative hypotension (IOH) by analyzing complex patterns in physiological waveforms such as arterial blood pressure (ABP), electroencephalogram (EEG), and electrocardiogram (ECG).

5.2.1 Analysis of Results

The achieved AUROC (Area Under the Receiver Operating Characteristic Curve) of 0.8679 and AUPRC (Area Under the Precision-Recall Curve) of 0.7759 indicate a high level of accuracy in the model’s predictive capability. These metrics suggest that the model is not only effective in distinguishing between instances where hypotension will and will not occur but also reliable in its probability estimations regarding the occurrence of hypotension.

- **AUROC**: An AUROC of 0.8679 demonstrates strong discriminative ability, particularly in a medical context where the cost of false negatives (failing to predict hypotension) can be critical.
- **AUPRC**: The AUPRC of 0.7759 further highlights the model’s effectiveness, especially considering that precision and recall are crucial in the clinical setting where both the precision of the positive predictions and the ability to capture as many positive instances as possible (recall) are important.

The sensitivity of 0.7507 and specificity of 0.8427, derived at an optimal threshold of 0.2311 based on the J-statistic, corroborate the model's robustness:

- **Sensitivity (True Positive Rate)**: The sensitivity rate indicates that the model successfully identifies 75.07% of all actual hypotensive events, which is vital for clinical applications where missing an event could lead to severe outcomes.
- **Specificity (True Negative Rate)**: The high specificity indicates that the model is also effective in correctly identifying when hypotension is not going to occur, avoiding unnecessary interventions.

5.2.2 Implications

These results validate the hypothesis and underscore the potential of using deep learning models to integrate and analyze complex physiological data for predicting IOH. The ability of the model to capture and learn from the nuanced dynamics of combined physiological signals confirms the hypothesis' premise and demonstrates the practical applicability of this approach in enhancing perioperative care. This could lead to improved patient monitoring, allowing for earlier interventions and potentially better patient outcomes.

Overall, the evidence from this experiment indicates that deep learning models, when properly trained on rich, multi-modal physiological data, provide a valuable tool for predicting intraoperative hypotension, thereby supporting the initial hypothesis and contributing positively to the field of predictive healthcare analytics.

5.3 Ablation Study

5.3.1 Additional Analysis: Using ABP and EEG Data Only

Overview

In this supplementary analysis, we focus on evaluating the performance of our predictive model utilizing only Arterial Blood Pressure (ABP) and Electroencephalogram (EEG) data, deliberately excluding Electrocardiogram (ECG) data. This approach aligns with the refined Hypothesis 2, which postulates that a model trained without ECG data might not only simplify the model but potentially improve or maintain the predictive accuracy for intraoperative hypotension (IOH). This analysis aims to explore the specific contributions of ECG data to the model's effectiveness and assess whether ABP and EEG alone can provide sufficient predictive power. The findings from this test could guide the development of more streamlined models in clinical settings, where computational efficiency and model simplicity are crucial.

Training code has been commented out (runtime: ~1hr)

In [22]:
```
"""
# Load the preprocessed dataset
data = np.load('preprocessed_data_200.npz')
x = data['x'][:, :, [0, 2]]  # Selecting only ABP and EEG channels
y = data['y']
valid_mask = data['valid_mask']
c = data['case_ids']

# Shuffle and split the dataset into training, validation, and testing sets
unique_cases = np.unique(c)
np.random.shuffle(unique_cases)

# Calculate split indices
n_cases = len(unique_cases)
n_train = int(n_cases * 0.6)
n_val = max(int(n_cases * 0.1), 1)
n_test = max(n_cases - n_train - n_val, 1)

# Define a function to extract data by cases
def extract_data(cases):
    indices = np.isin(c, cases)
    return x[indices], y[indices]

# Extract data subsets
x_train_abp_eeg, y_train_abp_eeg = extract_data(unique_cases[:n_train])
x_val_abp_eeg, y_val_abp_eeg = extract_data(unique_cases[n_train:n_train + n_val])
x_test_abp_eeg, y_test_abp_eeg = extract_data(unique_cases[n_train + n_val:])

# Save x_test_abp_eeg and y_test_abp_eeg as CSV files
np.savetxt("x_test_abp_eeg.csv", x_test_abp_eeg.reshape(x_test_abp_eeg.shape[0], -1), delimiter=",")
np.savetxt("y_test_abp_eeg.csv", y_test_abp_eeg, delimiter=",")

# Build the model specifically for ABP and EEG inputs
def build_abp_eeg_model(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = Conv1D(64, 8, strides=1, padding="same", kernel_regularizer=l2(1e-3))(inputs)
    x = BatchNormalization()(x)
    x = ReLU()(x)
    x = MaxPooling1D(pool_size=2)(x)
    x = GlobalAveragePooling1D()(x)
    outputs = Dense(num_classes, activation='sigmoid')(x)
    return Model(inputs=inputs, outputs=outputs)

model_abp_eeg = build_abp_eeg_model(x_train_abp_eeg.shape[1:], 1)
model_abp_eeg.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model_abp_eeg.summary()
"""
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 1500, 2)]         0

 conv1d_12 (Conv1D)          (None, 1500, 64)          1088

 batch_normalization_8 (Bat  (None, 1500, 64)          256
 chNormalization)

 re_lu_8 (ReLU)              (None, 1500, 64)          0

 max_pooling1d_2 (MaxPoolin  (None, 750, 64)           0
 g1D)

 global_average_pooling1d_2  (None, 64)                0
  (GlobalAveragePooling1D)
```

```
 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 1409 (5.50 KB)
Trainable params: 1281 (5.00 KB)
Non-trainable params: 128 (512.00 Byte)
_________________________________________________________________
```

In [23]:
```
"""
# Define the checkpoint path and filename
checkpoint_path = 'abp_eeg_model_checkpoint.h5'

# Create a ModelCheckpoint callback that saves the model's weights
checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=False,  # Set to True if you only want to save weights and not the entire model
    monitor='val_accuracy',   # Monitor validation accuracy for improvements
    save_best_only=True,      # Save only the best model
    verbose=1)                # Log the saving of the model
"""
```

In [24]:
```
"""
# Train the model with the ModelCheckpoint
history_abp_eeg = model_abp_eeg.fit(
    x_train_abp_eeg,
    y_train_abp_eeg,
    validation_data=(x_val_abp_eeg, y_val_abp_eeg),
    epochs=10,
    batch_size=32,
    callbacks=[checkpoint_callback],  # Add the callback here
    verbose=1)
"""
```

```
Epoch 1/10
2024-05-05 23:26:17.844110: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
4897/4897 [==============================] - ETA: 0s - loss: 0.4534 - accuracy: 0.8005
2024-05-05 23:28:06.213483: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
Epoch 1: val_accuracy improved from -inf to 0.78784, saving model to abp_eeg_model_checkpoint.h5
4897/4897 [==============================] - 118s 24ms/step - loss: 0.4534 - accuracy: 0.8005 - val_loss: 0.440
2 - val_accuracy: 0.7878
Epoch 2/10
4897/4897 [==============================] - ETA: 0s - loss: 0.4380 - accuracy: 0.8062
Epoch 2: val_accuracy improved from 0.78784 to 0.79533, saving model to abp_eeg_model_checkpoint.h5
4897/4897 [==============================] - 115s 24ms/step - loss: 0.4380 - accuracy: 0.8062 - val_loss: 0.435
4 - val_accuracy: 0.7953
Epoch 3/10
4897/4897 [==============================] - ETA: 0s - loss: 0.4350 - accuracy: 0.8084
Epoch 3: val_accuracy improved from 0.79533 to 0.79799, saving model to abp_eeg_model_checkpoint.h5
4897/4897 [==============================] - 117s 24ms/step - loss: 0.4350 - accuracy: 0.8084 - val_loss: 0.435
2 - val_accuracy: 0.7980
Epoch 4/10
4896/4897 [==============================>.] - ETA: 0s - loss: 0.4335 - accuracy: 0.8092
Epoch 4: val_accuracy did not improve from 0.79799
4897/4897 [==============================] - 117s 24ms/step - loss: 0.4335 - accuracy: 0.8092 - val_loss: 0.441
2 - val_accuracy: 0.7891
Epoch 5/10
4897/4897 [==============================] - ETA: 0s - loss: 0.4341 - accuracy: 0.8098
Epoch 5: val_accuracy did not improve from 0.79799
4897/4897 [==============================] - 116s 24ms/step - loss: 0.4341 - accuracy: 0.8098 - val_loss: 0.443
9 - val_accuracy: 0.7905
Epoch 6/10
4895/4897 [==============================>.] - ETA: 0s - loss: 0.4345 - accuracy: 0.8094
Epoch 6: val_accuracy did not improve from 0.79799
4897/4897 [==============================] - 116s 24ms/step - loss: 0.4345 - accuracy: 0.8094 - val_loss: 0.441
4 - val_accuracy: 0.7929
Epoch 7/10
4897/4897 [==============================] - ETA: 0s - loss: 0.4330 - accuracy: 0.8098
Epoch 7: val_accuracy did not improve from 0.79799
4897/4897 [==============================] - 115s 24ms/step - loss: 0.4330 - accuracy: 0.8098 - val_loss: 0.445
6 - val_accuracy: 0.7912
Epoch 8/10
4897/4897 [==============================] - ETA: 0s - loss: 0.4330 - accuracy: 0.8101
Epoch 8: val_accuracy improved from 0.79799 to 0.79935, saving model to abp_eeg_model_checkpoint.h5
4897/4897 [==============================] - 116s 24ms/step - loss: 0.4330 - accuracy: 0.8101 - val_loss: 0.434
9 - val_accuracy: 0.7993
Epoch 9/10
4896/4897 [==============================>.] - ETA: 0s - loss: 0.4309 - accuracy: 0.8108
Epoch 9: val_accuracy did not improve from 0.79935
4897/4897 [==============================] - 115s 23ms/step - loss: 0.4309 - accuracy: 0.8108 - val_loss: 0.439
0 - val_accuracy: 0.7941
```

```
6 - val_accuracy: 0.7941
Epoch 10/10
4896/4897 [============================>.] - ETA: 0s - loss: 0.4314 - accuracy: 0.8105
Epoch 10: val_accuracy improved from 0.79935 to 0.80381, saving model to abp_eeg_model_checkpoint.h5
4897/4897 [==============================] - 117s 24ms/step - loss: 0.4315 - accuracy: 0.8105 - val_loss: 0.436
6 - val_accuracy: 0.8038
```

In [53]:

```python
# URL of the model file on GitHub
url = 'https://github.com/yiboli1990/CS-598-DL4H-Project/raw/main/abp_eeg_model_checkpoint.h5'

# Load x_test_abp_eeg and y_test_abp_eeg from CSV files
x_test_abp_eeg = np.loadtxt("x_test_abp_eeg.csv", delimiter=",")
y_test_abp_eeg = np.loadtxt("y_test_abp_eeg.csv", delimiter=",")

# Reshape x_test_abp_eeg back to its original shape if necessary
x_test_abp_eeg = x_test_abp_eeg.reshape(-1, 1500, 2)  # Update dimensions for two channels: ABP and EEG

# Make a GET request to fetch the raw content by the URL
r = requests.get(url)
with open('downloaded_model_2.h5', 'wb') as f:
    f.write(r.content)

print("Model downloaded successfully.")

# Load the downloaded model
saved_model = load_model('downloaded_model_2.h5')

# Evaluate the model on the test set using the loaded model
test_loss_abp_eeg, test_acc_abp_eeg = saved_model.evaluate(x_test_abp_eeg, y_test_abp_eeg)
print(f"Test Accuracy for ABP and EEG model: {test_acc_abp_eeg * 100:.2f}%")

# Prediction and metrics calculation using the loaded model
y_prob_abp_eeg = saved_model.predict(x_test_abp_eeg)
auroc_abp_eeg = roc_auc_score(y_test_abp_eeg, y_prob_abp_eeg)
auprc_abp_eeg = average_precision_score(y_test_abp_eeg, y_prob_abp_eeg)

def calculate_sensitivity_specificity(y_true, y_scores, threshold):
    y_pred = (y_scores >= threshold).astype(int)
    cm = confusion_matrix(y_true, y_pred)
    sensitivity = cm[1, 1] / (cm[1, 0] + cm[1, 1])  # TP / (FN + TP)
    specificity = cm[0, 0] / (cm[0, 1] + cm[0, 0])  # TN / (FP + TN)
    return sensitivity, specificity

fpr_abp_eeg, tpr_abp_eeg, roc_thresholds_abp_eeg = roc_curve(y_test_abp_eeg, y_prob_abp_eeg)
J_abp_eeg = tpr_abp_eeg - fpr_abp_eeg
optimal_idx_abp_eeg = np.argmax(J_abp_eeg)
optimal_threshold_abp_eeg = roc_thresholds_abp_eeg[optimal_idx_abp_eeg]
sensitivity_abp_eeg, specificity_abp_eeg = calculate_sensitivity_specificity(y_test_abp_eeg, y_prob_abp_eeg, o

print("AUROC (Area Under the ROC Curve) for ABP and EEG model:", auroc_abp_eeg)
print("AUPRC (Area Under the Precision-Recall Curve) for ABP and EEG model:", auprc_abp_eeg)
print(f"Sensitivity for ABP and EEG model: {sensitivity_abp_eeg}")
print(f"Specificity for ABP and EEG model: {specificity_abp_eeg}")
print(f"Optimal threshold based on J-statistic for ABP and EEG model: {optimal_threshold_abp_eeg}")
```

```
Model downloaded successfully.
   1/2928 [..............................] - ETA: 11:33 - loss: 0.5921 - accuracy: 0.7812
2024-05-07 21:37:12.524415: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
2928/2928 [==============================] - 29s 10ms/step - loss: 0.4461 - accuracy: 0.8050
Test Accuracy for ABP and EEG model: 80.50%
  31/2928 [..............................] - ETA: 9s
2024-05-07 21:37:47.412635: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugi
n optimizer for device_type GPU is enabled.
2928/2928 [==============================] - 11s 4ms/step
AUROC (Area Under the ROC Curve) for ABP and EEG model: 0.8460344622277184
AUPRC (Area Under the Precision-Recall Curve) for ABP and EEG model: 0.7526750342042157
Sensitivity for ABP and EEG model: 0.7307833639552364
Specificity for ABP and EEG model: 0.8202030345346994
Optimal threshold based on J-statistic for ABP and EEG model: 0.33863362669944763
```

5.4 Conclusion for Hypothesis 2

The findings from the comparative analysis conducted to validate Hypothesis 2 provide a nuanced understanding of the role of ECG data in predicting intraoperative hypotension (IOH) when combined with ABP and EEG. Our experimental setup involved training two distinct sets of models: one harnessing all three physiological waveforms (ABP, EEG, ECG) and another exclusively using ABP and EEG data. This approach was designed to directly assess the impact of omitting ECG data on the predictive accuracy of our models.

5.4.1 Results Overview:

- **Model with ABP, EEG, and ECG:**
  - **AUROC**: 0.8679
  - **AUPRC**: 0.7759
  - **Sensitivity**: 0.7507
  - **Specificity**: 0.8427
  - **Optimal Threshold**: 0.2311
- **Model with ABP and EEG only:**
  - **AUROC**: 0.8460
  - **AUPRC**: 0.7527
  - **Sensitivity**: 0.7308
  - **Specificity**: 0.8202
  - **Optimal Threshold**: 0.3386

The analysis reveals that the exclusion of ECG data does not lead to a significant difference in the model's performance. The AUROC and AUPRC scores between the models are comparably high, indicating that ABP and EEG alone can match the predictive power of the model that also includes ECG data. Although there is a slight reduction in specificity and sensitivity, the overall results remain robust, underscoring the efficacy of ABP and EEG in capturing critical predictive signals for IOH.

5.4.2 Implications:

These results suggest that the integration of ECG data, does not fundamentally alter the model's capacity to predict IOH with high accuracy. This finding aligns with the observations from the original study, which suggested that models trained on just ABP and EEG were capable of achieving comparable or even slightly better performance than those that included ECG data. The marginal differences observed support the hypothesis that omitting ECG data might simplify the modeling process without meaningfully degrading performance. This could have significant implications for clinical settings where computational efficiency and model simplicity are paramount.

5.4.3 Concluding Remark:

In conclusion, the experimental results support Hypothesis 2 by demonstrating that the omission of ECG data does not meaningfully change the predictive accuracy of models designed to forecast intraoperative hypotension. These findings advocate for a potentially simpler and nearly equally effective approach to developing predictive models in healthcare, particularly in resource-constrained environments where model complexity may pose operational challenges.

5.5 Model Comparison

5.5.1 Comparative Discussion:

The results of our supplementary analysis, when compared with the original study, exhibit some differences that merit consideration. Below, we summarize key findings and possible explanations for the observed discrepancies:

Waveforms (3 min before event)	AUROC	AUPRC	Sensitivity	Specificity	Threshold
Original (ABP+ECG+EEG)	0.957	0.926	0.903	0.905	0.3
Original (ABP+EEG)	0.97	0.943	0.917	0.917	0.42
My Model (ABP+ECG+EEG)	0.868	0.776	0.751	0.843	0.23
My Model (ABP+EEG)	0.846	0.753	0.731	0.820	0.34

5.5.2 Possible Reasons for Performance Differences:

1. **Data Volume**: The original study utilized a larger dataset, which typically contributes to more robust and generalizable model training. More data can help the model learn more comprehensive patterns and reduce overfitting.
2. **Waveform Quality**: The original research included a method to exclude low-quality waveform data, enhancing the reliability of the signals used for training the model. Cleaner data generally leads to better model performance.
3. **Sampling Rate**: The original paper used a higher sampling rate (500 Hz), which provides more granular and detailed signal information than the reduced sampling rate (25 Hz) used in our experiments. Higher resolution data can capture subtle changes in physiological signals that are crucial for accurate predictions.

4. **Feature Engineering**: There may be differences in how features were engineered and extracted from the waveforms. Advanced or more appropriate feature engineering techniques can significantly impact model performance.
5. **Model Architectural Differences**: Minor differences in the neural network architecture, hyperparameters, or training procedures could also account for variations in performance.

5.5.3 Conclusion:

While our model shows promising results, aligning our experimental setup more closely with that of the original study, including using higher-quality and higher-resolution data, could potentially enhance our model's predictive accuracy. This analysis underscores the importance of data quality, volume, and detailed feature extraction in developing effective predictive models in healthcare.

6 Discussion

The attempt to reproduce the findings from the original paper on predicting intraoperative hypotension (IOH) using physiological waveform data has yielded insightful results, albeit with some deviations. This section evaluates the reproducibility of the original study, discusses the challenges encountered, and outlines future plans.

6.1 Reproducibility Assessment

Based on the results obtained from our experiments, the paper is partially reproducible. We were able to implement similar models and achieve comparable trends in predictive performance metrics, such as AUROC and AUPRC. However, our results did not entirely match the precision of the outcomes reported in the original study. This discrepancy can largely be attributed to differences in data volume and quality, as well as variations in computational resources.

6.2 Challenges in Reproducibility

- **What was easy**:
  - **Model Implementation**: Building the neural network architecture and training it with TensorFlow/Keras was straightforward due to the well-documented nature of these libraries and the clear model specifications provided in the original paper.
  - **Data Access**: Accessing the VitalDB dataset for physiological waveforms was uncomplicated, which facilitated the initial steps of data handling and preprocessing.
- **What was difficult**:
  - **Data Quality Control**: The original study included mechanisms to exclude low-quality waveform data which we could not fully replicate due to limited access to sophisticated preprocessing tools and constraints in computational resources.
  - **High Sampling Rates**: Replicating models trained on high-resolution data (500 Hz in the original study) was challenging with our reduced sampling rate (25 Hz), potentially affecting the granularity and the accuracy of predictions.
  - **Volume of Data**: The original paper utilized a significantly larger dataset, which likely contributed to the robustness and accuracy of their model. Our smaller dataset might not capture as diverse a range of physiological variations.