알츠하이머병 분류를 위한 TinyCNN 모델 사용하기

이번 노트북에서는 PyTorch를 사용하여 TinyCNN 모델을 통해 알츠하이머 MRI 질병 분류 데이터셋의 MRI 이미지를 분류할 것입니다. 이 데이터셋은 네 가지 범주로 나뉘어 있습니다:

- **비치매 (Non-Demented): 건강한 뇌
- **매우 경미한 치매 (Very Mild Demented): 초기 치매 증상
- **경미한 치매 (Mild Demented): 경미한 인지 저하
- **중등도 치매 (Moderate Demented): 중등도 치매 증상

우리의 목표는 이러한 단계들을 구분할 수 있는 모델을 구축하여 알츠하이머병의 조기 진단을 지원하는 것입니다.

단계:

1. **TinyCNN 구축**: 효율적인 이미지 분류를 위한 경량 CNN 설계.
2. **데이터셋 준비**: MRI 이미지 로드 및 전처리.
3. **모델 훈련**: TinyCNN이 치매 단계를 식별하도록 학습.
4. **성능 평가**: 정확도 및 손실 측정.
5. **예측 시각화**: 샘플 이미지에 대한 모델 예측 확인.

Let's get started and see how well our model can support Alzheimer's detection!

알츠하이머 MRI 질병 분류 데이터셋

이 데이터셋은 연구자와 의료 분야에 유용한 자원으로, 알츠하이머 질병을 MRI 스캔을 기반으로 분류하는 데 중점을 두고 있습니다. 데이터셋은 뇌 MRI 이미지를 네 가지 범주로 레이블링하여 구성됩니다:

1. '0': 경미한 치매 (Mild_Demented)
2. '1': 중등도 치매 (Moderate_Demented)
3. '2': 비치매 (Non_Demented)
4. '3': 매우 경미한 치매 (Very_Mild_Demented)

데이터셋 정보

훈련 데이터 (Train split):

이름: train 바이트 수: 22,560,791.2 예제 수: 5,120 테스트 데이터 (Test split):

이름: test 바이트 수: 5,637,447.08 예제 수: 1,280 총 다운로드 크기: 28,289,848 바이트

총 데이터셋 크기: 28,198,238.28 바이트

Source https://huggingface.co/datasets/Falah/Alzheimer_MRI

Citation If you use this dataset in your research or health medicine applications, we kindly request that you cite the following publication:

In [2]:
```
import torch
import tqdm
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
from torch.utils.data import Dataset, DataLoader
import numpy as np
import random
import pandas as pd
import seaborn as sns
import cv2
import torchvision
from torchvision import datasets, models, transforms
from torchvision.transforms import v2
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import time
import os
import PIL
from PIL import Image
from collections import Counter
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
from tqdm import tqdm
from torch.utils.data import DataLoader, random_split
from PIL import Image, ImageEnhance
import numpy as np
import matplotlib.pyplot as plt
import torch
from torchvision import transforms


cudnn.benchmark = True

device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

```
Using cpu device
```

In [3]:
```
BASE_DIR = "Alzheimer_s Dataset.v1i.folder"
disease_label_from_category = {
    0: "Mild Demented",
    1: "Moderate Demented",
    2: "Non Demented",
    3: "Very Mild Demented",
}
```

Reading Data

데이터는 Parquet 파일 형식으로 저장되어 있습니다. Pandas를 사용하여 간단히 읽을 수 있습니다.

In [7]:
```
df = pd.read_parquet(f"Alzheimer_s Dataset.v1i.folder/Data/train-00000-of-00001-c08a401c53fe5312.parquet", engi
df.head()
```

Out[7]:

	image	label
0	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	2
1	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	0
2	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	3
3	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	3
4	{'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x...	2

In [9]:
```
test = pd.read_parquet(f"Alzheimer_s Dataset.v1i.folder/Data/train-00000-of-00001-c08a401c53fe5312.parquet", en
```

이 데이터는 특이한 형식으로 되어 있음을 알 수 있습니다. 이는 단순히 이미지와 레이블을 포함하는 사전 형태입니다. 우리가 가장 먼저 해야 할 일은 이러한 암호화된 "이미지"를 픽셀의 강도를 나타내는 NumPy 배열로 변환하는 것입니다(우리는 MRI가 그레이스케일이라고 가정합니다).

In [12]:
```
def dict_to_image(image_dict):
    if isinstance(image_dict, dict) and 'bytes' in image_dict:
        byte_string = image_dict['bytes']
        nparr = np.frombuffer(byte_string, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_GRAYSCALE)
        return img
    else:
        raise TypeError(f"Expected dictionary with 'bytes' key, got {type(image_dict)}")
```

In [14]:
```
df['img_arr'] = df['image'].apply(dict_to_image)
df.drop("image", axis=1, inplace=True)
df.head()
```

Out[14]:

	label	img_arr
0	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
1	0	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
2	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
3	3	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
4	2	[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...

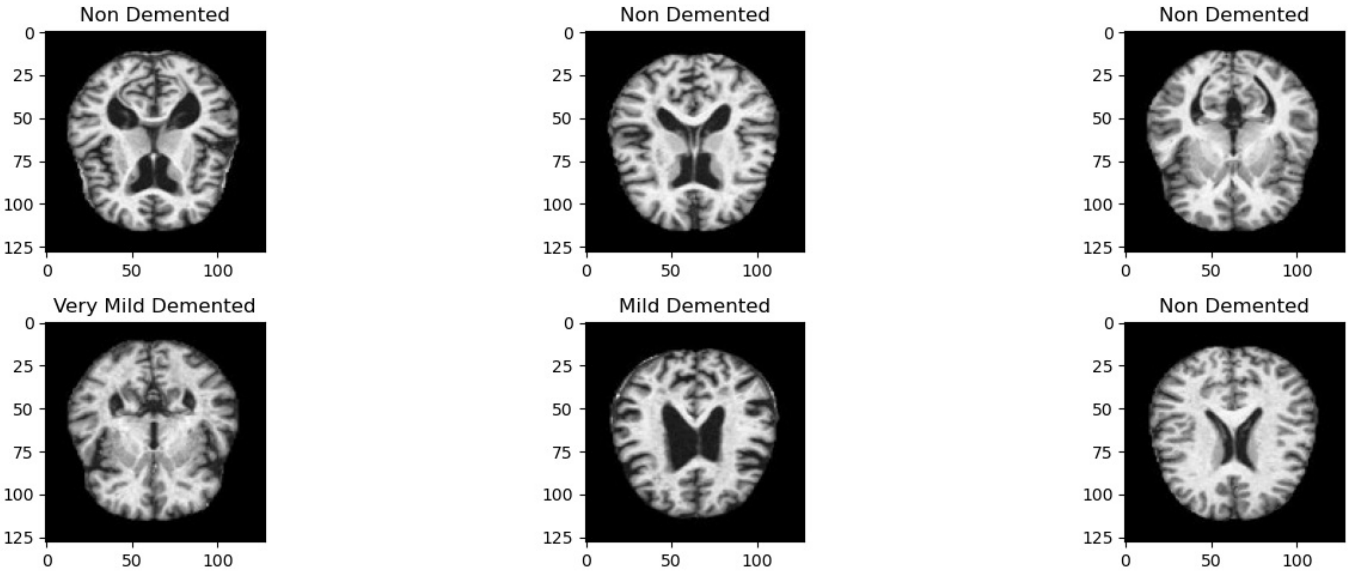
In [15]:
```
test['img_arr'] = test['image'].apply(dict_to_image)
```

```
test.drop("image", axis=1, inplace=True)
```

변환이 완료되었으므로, 몇 개의 이미지를 시각화하여 모든 것이 올바르게 작동했는지 확인할 수 있습니다. 또한, 모델을 나중에 올바르게 설정하기 위해 이미지의 형태를 이해해야 합니다.

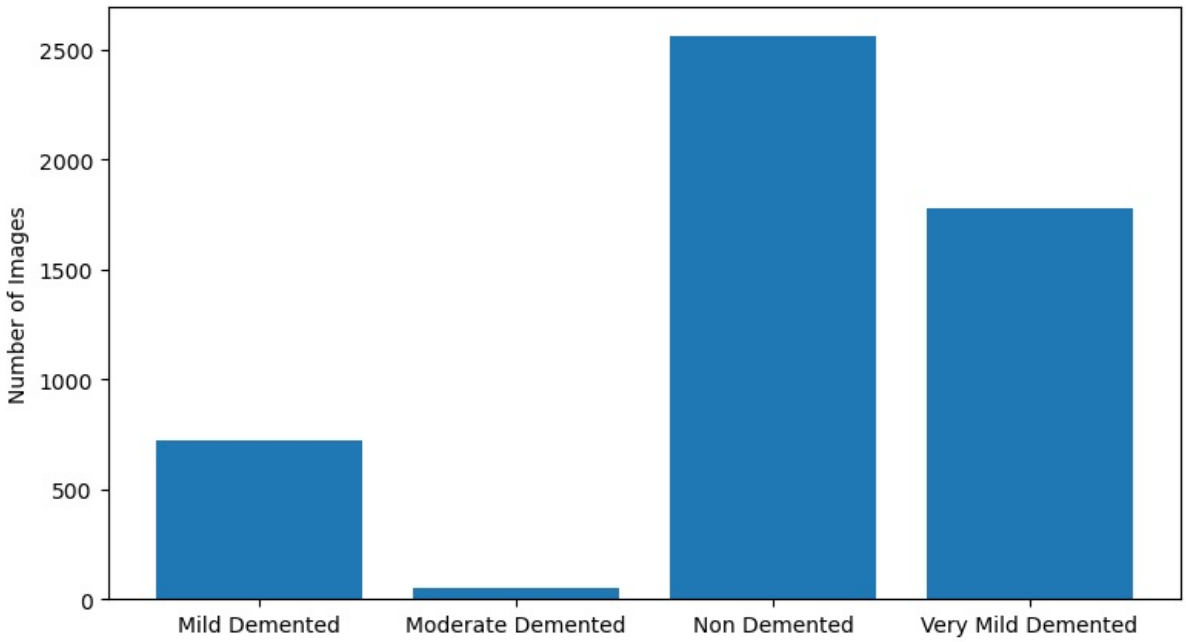
Exploring Data

In [20]:
```
# Check we can actually render the image and that it looks reasonable
fig, ax = plt.subplots(2, 3, figsize=(15, 5))
axs = ax.flatten()
for axes in axs:
    rand = np.random.randint(0, len(df))
    axes.imshow(df.iloc[rand]['img_arr'], cmap="gray")
    axes.set_title(disease_label_from_category[df.iloc[rand]['label']])
plt.tight_layout()
plt.show()
```



- **데이터 분포 분석**: 각 클래스(비치매, 매우 경미한 치매, 경미한 치매, 중등도 치매)에 대한 이미지 수를 확인하여 데이터의 균형 상태를 파악합니다.
- **샘플 이미지 시각화**: 각 클래스에서 몇 가지 샘플 이미지를 시각화하여 데이터의 특성을 이해합니다. 이를 통해 이미지의 다양성과 특징을 파악할 수 있습니다.
- **이미지 크기 및 형식 확인**: MRI 이미지의 크기와 채널 수를 확인하여 모델의 입력 형태를 결정합니다.
- **데이터 증강 적용**: 훈련 데이터를 다양화하기 위해 회전, 이동, 크기 조정, 플립 등의 데이터 증강 기법을 적용하여 모델의 일반화 능력을 향상시킬 수 있습니다.

In [23]:
```
plt.figure(figsize=(9, 5))
plt.bar(np.arange(0, 4, 1), df['label'].value_counts().sort_index())
plt.ylabel("Number of Images")
plt.xticks(np.arange(0, 4, 1), labels=[disease_label_from_category[i] for i in range(4)])
plt.show()
print(f"Total samples in training data = {len(df)}")
```




```
Total samples in training data = 5120
```

이 플롯에서 확인할 수 있는 것은 우리 데이터셋에 "중등도" 심각도의 사례가 거의 없다는 점입니다. 따라서 모델이 이러한 사례를 식별하는 데 가장 약할 것으로 예상됩니다. 이는 배울 수 있는 데이터가 매우 적기 때문입니다.

또한 데이터셋에는 약 5000개의 이미지가 있다는 점도 관찰할 수 있습니다. 이는 자체적으로 꽤 많은 양이지만, 데이터 증강을 통해 통계치를 향상시킬 수 있습니다! 이는 경미한 및 중등도 치매 환자의 MRI 스캔이 훨씬 적기 때문에 매우 유용할 것입니다.

Step 1: 데이터 로드 및 레이블을 이진 클래스로 변환하기

이 단계에서는 이미지 배열과 레이블을 포함한 .parquet 파일에서 MRI 이미지를 로드합니다. 분류를 단순화하기 위해 데이터셋을 이진 문제로 변환할 것입니다:

- **비치매 (건강한 뇌)**:: Label as **0**.
- **치매**: Label as **1**, 매우 경미한, 경미한, 중등도 치매 단계를 통합합니다.

다음 코드는 .parquet 데이터를 로드하고 이러한 이진 레이블을 적용하는 작업을 수행

In [27]:
```python
# Define binary label mapping based on dataset description
label_mapping = {
    0: 1,  # Mild_Demented -> Demented
    1: 1,  # Moderate_Demented -> Demented
    2: 0,  # Non_Demented -> Healthy
    3: 1   # Very_Mild_Demented -> Demented
}

# Custom Dataset class for Alzheimer's MRI binary classification with .parquet
class AlzheimerBinaryDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        # Load the .parquet file into a DataFrame
        self.dataframe = dataframe

         # Check unique values in the 'label' column before mapping
        print("Unique labels before mapping:", self.dataframe["label"].unique())

        # Apply binary label mapping to the 'label' column
        self.dataframe["label"] = self.dataframe["label"].map(label_mapping)

        # Verify label mapping (should only contain 0 and 1)
        unique_labels = self.dataframe["label"].unique()
        if not all(label in [0, 1] for label in unique_labels):
            raise ValueError(f"Unexpected label values found after mapping: {unique_labels}")

        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        # Load the image array and binary label from the DataFrame
        image = self.dataframe.iloc[idx]["img_arr"]
        label = self.dataframe.iloc[idx]["label"]

        # Convert image to tensor, add channel dimension, and apply transformations if any
        image = torch.tensor(image, dtype=torch.float32).unsqueeze(0)  # Shape (1, height, width)
```

```
        label = torch.tensor(label, dtype=torch.long)  # Ensure label is long type for classification

        if self.transform:
            image = self.transform(image)

        return image, label
```

In [29]:
```
# Define any additional transforms here if needed
# Define the transformation: Resize to 28x28, normalize, and add channel dimension
transform = transforms.Compose([
    transforms.Resize((28, 28)),             # Resize to 28x28 pixels
    transforms.Normalize((0.5,), (0.5,))     # Normalize pixel values to range [-1, 1]
])

# Initialize the dataset and DataLoader
binary_dataset = AlzheimerBinaryDataset(df, transform=transform)

# Check a sample from the dataset
sample_image, sample_label = binary_dataset[0]
print("Sample image shape:", sample_image.shape)  # Should be (1, 128, 128) or similar
print("Sample label (binary):", sample_label)  # Should be 0 or 1
```

```
Unique labels before mapping: [2 0 3 1]
Sample image shape: torch.Size([1, 28, 28])
Sample label (binary): tensor(0)
```

Step 2: 훈련 및 검증 DataLoader 설정하기

PyTorch의 **DataLoader**는 알츠하이머 MRI 데이터셋을 미니 배치로 효율적으로 로드하여 훈련 및 검증에 필수적입니다.

주요 설정 사항:

- **batch_size**: 배치당 이미지 수 (예: 32).
- **shuffle**: 훈련 데이터를 무작위로 섞어 일반화를 개선; 검증 데이터에는 적용되지 않음.
- **num_workers**: 데이터 로드를 위한 서브 프로세스 수를 설정합니다..

DataLoader 설정:

- **훈련 세트**: 모델을 훈련하고 패턴을 학습하는 데 사용됩니다.
- **검증 세트**: 각 에포크 후 성능과 모델의 일반화를 평가하는 데 사용됩니다.

In [32]:
```
# Define the train-validation split ratio
train_ratio = 0.8  # 80% for training, 20% for validation

# Calculate the number of samples in each subset
train_size = int(train_ratio * len(binary_dataset))
val_size = len(binary_dataset) - train_size

# Split the dataset
train_dataset, val_dataset = random_split(binary_dataset, [train_size, val_size])

# Create DataLoaders for the train and validation subsets
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

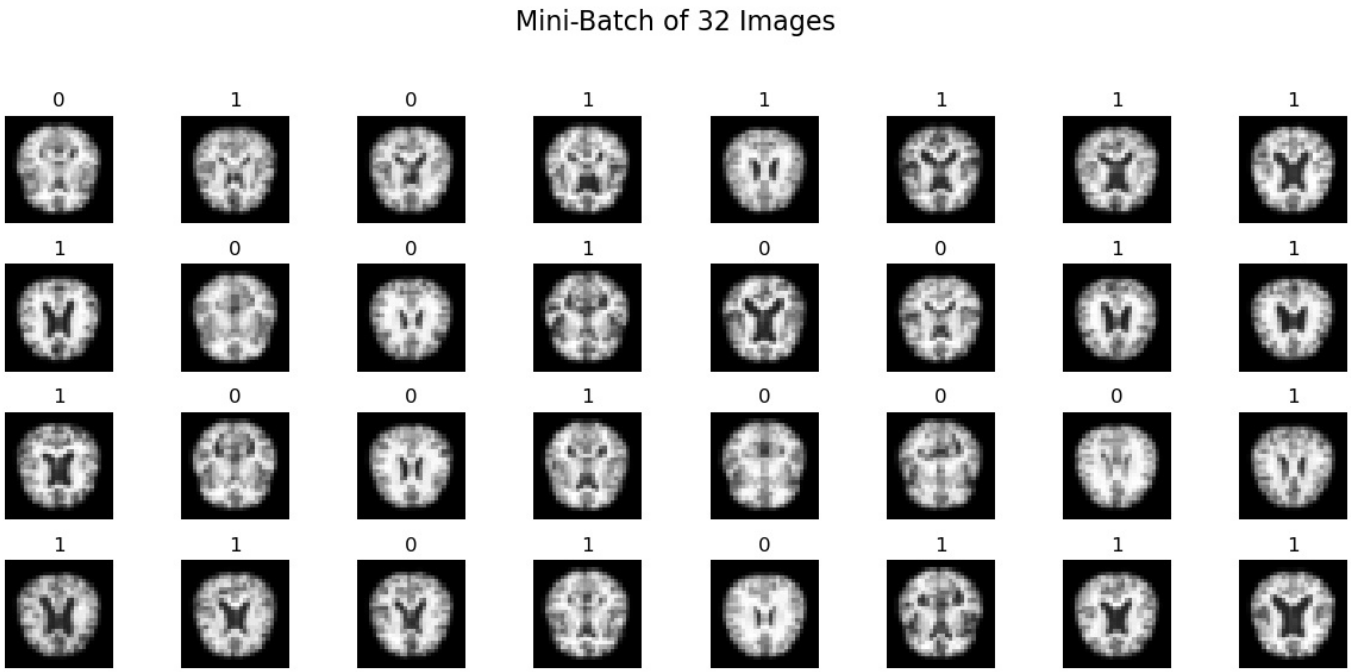
Step 3: 이미지 배치 시각화하기

알츠하이머 MRI 데이터셋에 대한 통찰을 얻기 위해, 이진 레이블(**0은 비치매, 1은 치매**)과 함께 미니 배치의 이미지를 시각화할 것입니다.

이미지는 4x8 그리드로 표시되며, 각 이미지는 **0** (**비치매**) 또는 **1** (**치매**)로 레이블이 붙습니다. 이 시각화를 통해 데이터 로드, 변환, 그리고 이진 레이블링을 확인할 수 있습니다.

In [35]:
```
# Get a single batch of images and labels
batch_images, batch_labels = next(iter(train_loader))

# Display the batch of images
fig, axes = plt.subplots(4, 8, figsize=(12, 6))
fig.suptitle(f"Mini-Batch of {len(batch_images)} Images", fontsize=16)

# Loop through each image in the batch and display it
for i, (img, label) in enumerate(zip(batch_images[:32], batch_labels[:32])):  # Limit to 32 images for display
    ax = axes[i // 8, i % 8]
    img = img.squeeze().numpy()  # Convert from Tensor format to NumPy array for display
    img = img * 0.5 + 0.5  # De-normalize for visualization (if normalization was applied)
    ax.imshow(img, cmap="gray")
    ax.set_title(str(label.item()))  # Display binary label (0 or 1)
    ax.axis("off")
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

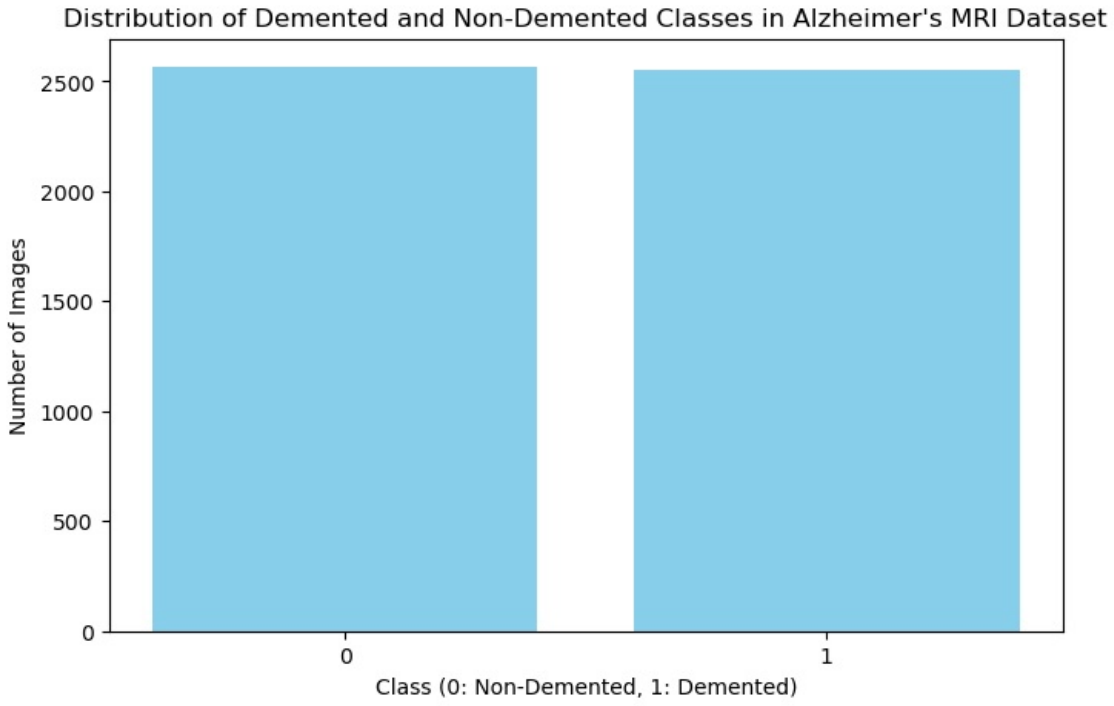


Step 4: 알츠하이머 MRI 데이터셋의 데이터 분포 확인하기

훈련 전에 알츠하이머 MRI 데이터셋의 클래스 분포를 조사하여 균형 잡힌 데이터셋인지 확인합니다. 데이터 분포를 이해하는 것은 모델 성능에 영향을 줄 수 있는 클래스 불균형을 식별하는 데 도움이 됩니다.

1. **클래스별 이미지 수 세기**: 이진 설정에서 각 카테고리의 이미지 수를 세겠습니다:
   - **0**: Non-Demented (Healthy)
   - **1**: Demented (combining Very Mild, Mild, and Moderate Demented)
2. **분포 시각화**: 클래스 분포를 플롯하여 클래스가 고르게 표현되고 있는지 확인합니다.

In [37]:
```
# Count the number of images for each class in the dataset
class_counts = Counter(binary_dataset.dataframe["label"])

# Extract class labels and counts
classes = list(class_counts.keys())
counts = list(class_counts.values())

# Plot the class distribution
plt.figure(figsize=(8, 5))
plt.bar(classes, counts, color='skyblue')
plt.xlabel("Class (0: Non-Demented, 1: Demented)")
plt.ylabel("Number of Images")
plt.title("Distribution of Demented and Non-Demented Classes in Alzheimer's MRI Dataset")
plt.xticks(classes)
plt.show()
```



Step 5: 알츠하이머 MRI 분류를 위한 TinyCNN 구축 및 초기화

알츠하이머 MRI 이진 분류 작업을 위해 커스텀 TinyCNN 모델을 사용할 것입니다. 이 모델은 경량이며 효율적으로 설계되어 작은 데이터셋에서 빠르게 훈련할 수 있어, 신속한 시연 및 실험에 적합합니다.

1. **TinyCNN 아키텍처 정의**: 이 모델은 필수 이미지 특징을 캡처하기 위한 최소한의 합성곱 및 풀링 레이어를 포함하며, 이진 분류(비치매 vs. 치매)를 위한 완전 연결층이 뒤따릅니다.
2. **모델 초기화**: 훈련 가능한 매개변수 수를 낮게 유지하기 위해 Compact Layer로 TinyCNN을 설정합니다.
3. **출력 클래스 지정**: 마지막 레이어는 2개의 클래스를 출력합니다 (0은 비치매, 1은 치매).
4. **모델을 디바이스로 이동**: GPU가 사용 가능한 경우 모델을 GPU로 전송하고, 그렇지 않으면 CPU로 전송하여 빠른 처리를 가능하게 합니다.

이 설정을 통해 알츠하이머 MRI 데이터셋에서 빠르게 훈련하면서 치매 분류를 위한 의미 있는 결과를 달성할 수 있습니다.

In [41]:
```python
# Define a Depthwise Separable Convolutional Layer
class DepthwiseSeparableConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, stride=1, padding=1):
        super(DepthwiseSeparableConv, self).__init__()
        # Depthwise convolution
        self.depthwise = nn.Conv2d(in_channels, in_channels, kernel_size=kernel_size,
                                   stride=stride, padding=padding, groups=in_channels)
        # Pointwise convolution
        self.pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        x = self.depthwise(x)
        x = self.pointwise(x)
        return x

# Define the TinyCNN model using Depthwise Separable Convolutions for Alzheimer's MRI
class TinyCNN(nn.Module):
    def __init__(self):
        super(TinyCNN, self).__init__()
        # Use Depthwise Separable Convolutions for efficient computation
        self.conv1 = DepthwiseSeparableConv(1, 16, kernel_size=3, stride=1, padding=1)  # 1 input channel, 16 ou
        self.conv2 = DepthwiseSeparableConv(16, 32, kernel_size=3, stride=1, padding=1)  # 16 input channels, 32
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Fully connected layer with 2 output classes for binary classification
        self.fc1 = nn.Linear(32 * 7 * 7, 2)  # Output layer for 2 classes (0: Non-Demented, 1: Demented)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # Conv1 + ReLU + Pooling
        x = self.pool(F.relu(self.conv2(x)))  # Conv2 + ReLU + Pooling
        x = x.view(-1, 32 * 7 * 7)  # Flatten for fully connected layer
        x = self.fc1(x)  # Output layer
        return x

# Initialize the model
```

```
model = TinyCNN()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

print("Model architecture:")
print(model)
```

```
Model architecture:
TinyCNN(
  (conv1): DepthwiseSeparableConv(
    (depthwise): Conv2d(1, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (pointwise): Conv2d(1, 16, kernel_size=(1, 1), stride=(1, 1))
  )
  (conv2): DepthwiseSeparableConv(
    (depthwise): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16)
    (pointwise): Conv2d(16, 32, kernel_size=(1, 1), stride=(1, 1))
  )
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1568, out_features=2, bias=True)
)
```

In [43]:
```
# Check number of parameters
total_params = sum(p.numel() for p in model.parameters())
print(f"Total parameters: {total_params}")
```

```
Total parameters: 3884
```

Step 6: 손실 함수 및 최적화기 정의하기

우리의 MobileNetV3 모델을 훈련하기 위해서는 다음이 필요합니다:

1. **손실 함수** (Criterion): CrossEntropyLoss는 모델의 예측이 실제 레이블과 얼마나 잘 일치하는지를 측정합니다.
2. **최적화기**: Adam 최적화기는 계산된 그래디언트를 기반으로 모델의 가중치를 업데이트하는 데 도움을 줍니다.

In [46]:
```
# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Step 7: TinyCNN 훈련 및 평가하기

이제 알츠하이머 MRI 데이터셋에서 TinyCNN 모델을 이진 분류(0: 비치매, 1: 치매)로 훈련할 것입니다. 훈련 루프는 여러 에포크에 걸쳐 반복되며, 모델의 매개변수를 최적화하여 훈련 세트의 손실을 최소화합니다. 각 에포크 후에는 검증 세트에서 모델을 평가하여 보지 않은 데이터에 대한 성능을 모니터링합니다.

훈련과정개요:

1. **훈련 단계**: 각 미니 배치에 대해 이진 크로스 엔트로피 손실을 계산하고, 역전파를 수행하여 모델 매개변수를 업데이트합니다. 또한 훈련 정확도를 기록합니다.
2. **검증 단계**: 각 훈련 에포크 후, 검증 세트에서 모델을 평가하여 비치매와 치매 MRI 스캔을 구별하는 일반화 성능을 측정합니다.
3. **지표**: 각 에포크 후 훈련 손실, 훈련 정확도, 검증 손실 및 검증 정확도를 출력하여 모델의 진행 상황을 추적합니다.

Let's begin training!

In [49]:
```
# Hyperparameters
num_epochs =10
learning_rate = 0.001

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()  # Suitable for binary classification with class labels 0 and 1
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training and validation loop
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    # Training phase
    model.train()
    running_loss, correct, total = 0.0, 0, 0

    for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} - Training"):
        images, labels = images.to(device), labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
```

```
        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Update metrics
        running_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_loss = running_loss / total
    train_accuracy = correct / total
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    # Validation phase
    model.eval()
    val_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Update metrics
            val_loss += loss.item() * images.size(0)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

    val_loss /= total
    val_accuracy = correct / total
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # Print metrics after each epoch
    print(f"Epoch [{epoch+1}/{num_epochs}]")
    print(f"Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}")
    print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}")
    print("-" * 30)
```

```
Epoch 1/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 55.51it/s]
Epoch [1/10]
Train Loss: 1.0645, Train Accuracy: 0.5925
Validation Loss: 0.6076, Validation Accuracy: 0.6914
------------------------------
Epoch 2/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 55.20it/s]
Epoch [2/10]
Train Loss: 0.6734, Train Accuracy: 0.6562
Validation Loss: 0.5858, Validation Accuracy: 0.7178
------------------------------
Epoch 3/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 57.40it/s]
Epoch [3/10]
Train Loss: 0.6038, Train Accuracy: 0.6924
Validation Loss: 0.5588, Validation Accuracy: 0.7393
------------------------------
Epoch 4/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 54.80it/s]
Epoch [4/10]
Train Loss: 0.5527, Train Accuracy: 0.7317
Validation Loss: 0.7509, Validation Accuracy: 0.6494
------------------------------
Epoch 5/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 57.32it/s]
Epoch [5/10]
Train Loss: 0.5606, Train Accuracy: 0.7185
Validation Loss: 0.5153, Validation Accuracy: 0.7646
------------------------------
Epoch 6/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 56.22it/s]
Epoch [6/10]
Train Loss: 0.4673, Train Accuracy: 0.7783
Validation Loss: 0.5256, Validation Accuracy: 0.7363
------------------------------
```

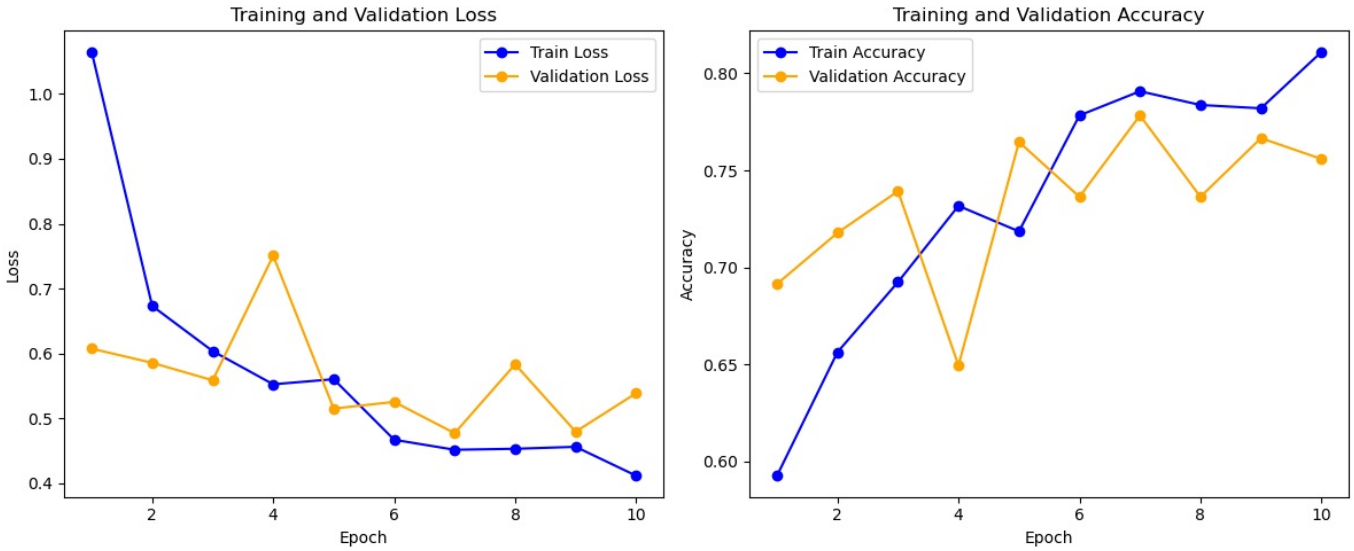
```
Epoch 7/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 57.68it/s]
Epoch [7/10]
Train Loss: 0.4518, Train Accuracy: 0.7908
Validation Loss: 0.4774, Validation Accuracy: 0.7783
------------------------------
Epoch 8/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 54.75it/s]
Epoch [8/10]
Train Loss: 0.4534, Train Accuracy: 0.7837
Validation Loss: 0.5835, Validation Accuracy: 0.7363
------------------------------
Epoch 9/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 56.84it/s]
Epoch [9/10]
Train Loss: 0.4564, Train Accuracy: 0.7820
Validation Loss: 0.4797, Validation Accuracy: 0.7666
------------------------------
Epoch 10/10 - Training: 100%|██████████| 128/128 [00:02<00:00, 58.05it/s]
Epoch [10/10]
Train Loss: 0.4119, Train Accuracy: 0.8110
Validation Loss: 0.5390, Validation Accuracy: 0.7559
------------------------------
```

훈련에서 100% 정확도를 기록했다면, 우리가 과적합(overfitting)된 가능성이 큽니다..

Step 8: 훈련 및 검증 지표 시각화하기

에포크에 따른 손실과 정확도를 시각화하면 모델이 얼마나 잘 학습하고 있는지를 이해하는 데 도움이 됩니다. 이상적으로는 훈련 손실과 검증 손실 모두 감소하고, 정확도는 증가해야 합니다.

In [53]:
```python
# Plot Loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss', color='blue', marker='o')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss', color='orange', marker='o')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.title("Training and Validation Loss")

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy', color='blue', marker='o')
plt.plot(range(1, num_epochs + 1), val_accuracies, label='Validation Accuracy', color='orange', marker='o')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Training and Validation Accuracy")

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



주어진 그래프는 모델의 훈련 과정에서의 손실(loss)과 정확도(accuracy)를 나타내며, 훈련 데이터와 검증 데이터의 성능 변화를 보여줍니다. 왼쪽 그래프는 손실에 대한 것이고, 오른쪽 그래프는 정확도에 대한 것입니다. 각 그래프를 해석해보겠습니다.

왼쪽 그래프: Training and Validation Loss

- Y축 (Loss): 손실 값으로, 값이 낮을수록 모델의 성능이 좋음을 나타냅니다.
- X축 (Epoch): 에폭 수로, 훈련이 진행된 횟수를 나타냅니다.
- 파란색 선 (Train Loss): 훈련 데이터에 대한 손실입니다. 에폭이 증가함에 따라 손실 값이 감소하고 있음을 보여줍니다. 이는 모델이 훈련 데이터에 잘 적합하고 있다는 것을 의미합니다.
- 주황색 선 (Validation Loss): 검증 데이터에 대한 손실입니다. 훈련 손실과 유사하게 감소하고 있으나, 몇몇 에폭에서 약간의 변동이 있습니다. 이는 모델이 검증 데이터에서도 잘 작동하고 있다는 것을 의미합니다.
- 결론: 훈련 손실과 검증 손실 모두 감소하는 추세를 보이며, 모델이 적절하게 학습되고 있음을 나타냅니다. 훈련 손실이 검증 손실보다 낮은 경우 일반적인 현상이며, 이는 모델이 훈련 데이터에 과적합(overfitting)되지 않고 있다는 것을 시사합니다.

오른쪽 그래프: Training and Validation Accuracy

- Y축 (Accuracy): 정확도로, 값이 높을수록 모델의 성능이 좋음을 나타냅니다.
- X축 (Epoch): 에폭 수로, 훈련이 진행된 횟수를 나타냅니다.
- 파란색 선 (Train Accuracy): 훈련 데이터에 대한 정확도입니다. 에폭이 증가함에 따라 훈련 정확도가 꾸준히 증가하고 있음을 보여줍니다.
- 주황색 선 (Validation Accuracy): 검증 데이터에 대한 정확도입니다. 훈련 정확도와 유사하게 증가하지만, 몇몇 에폭에서 약간의 변동이 있습니다. 이는 검증 데이터에서도 모델이 잘 작동하고 있음을 나타냅니다.
- 결론: 훈련 정확도와 검증 정확도 모두 증가하는 추세를 보이며, 모델이 학습하고 있다는 것을 나타냅니다. 훈련 정확도가 검증 정확도보다 높지만, 두 값 모두 비슷한 추세를 보이므로 과적합의 위험이 크지 않다고 볼 수 있습니다.

종합 해석 전체적으로 모델은 훈련 과정에서 손실을 줄이고 정확도를 높이며 잘 학습되고 있습니다. 훈련 손실과 검증 손실 모두 감소하고, 훈련 정확도와 검증 정확도 모두 증가하는 추세를 보여 과적합의 위험이 낮습니다. 모델의 성능을 더욱 개선하기 위해 추가적인 에폭을 진행하거나, 하이퍼파라미터 조정 및 데이터 증강(data augmentation) 등의 방법을 고려할 수 있습니다.

Step 10: 랜덤 테스트 이미지 예측하기

모델의 성능을 랜덤 테스트 이미지에서 평가하기 위해 다음과 같은 과정을 진행합니다:

1. **랜덤 이미지 선택**: 테스트 세트에서 랜덤 샘플을 선택합니다.
2. **전처리 및 예측**: 이미지를 전처리한 후 모델을 통해 통과시키고, 출력을 확률로 변환합니다..
3. **결과 표시**: 예측된 레이블, 신뢰도 수준, 실제 레이블과 함께 이미지를 보여줍니다

결과 표시: 예측된 레이블, 신뢰도 수준, 실제 레이블과 함께 이미지를 보여줍니다

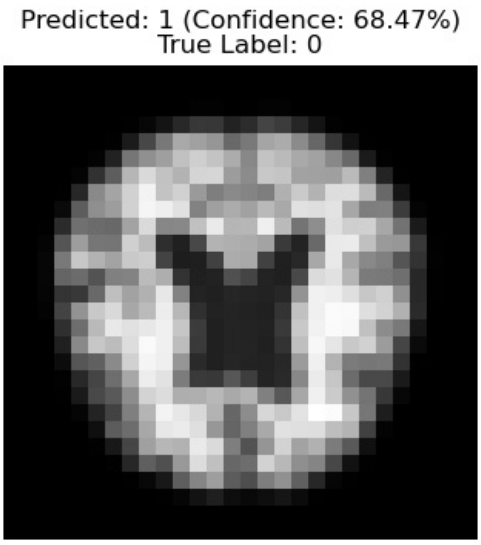
In [57]:
```
# Define any additional transforms here if needed
# Define the transformation: Resize to 28x28, normalize, and add channel dimension
transform = transforms.Compose([
    transforms.Resize((28, 28)),             # Resize to 28x28 pixels
    transforms.Normalize((0.5,), (0.5,))     # Normalize pixel values to range [-1, 1]
])

# Initialize the dataset and DataLoader
test_dataset = AlzheimerBinaryDataset(test, transform=transform)

# Check a sample from the dataset
sample_image, sample_label = test_dataset[0]
print("Sample image shape:", sample_image.shape)  # Should be (1, 128, 128) or similar
print("Sample label (binary):", sample_label)  # Should be 0 or 1
```

```
Unique labels before mapping: [2 0 3 1]
Sample image shape: torch.Size([1, 28, 28])
Sample label (binary): tensor(0)
```

In [59]:
```
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

In [61]:
```
# Choose a random image from the test set
random_index = random.randint(0, len(test_dataset) - 1)
random_img, random_label = test_dataset[random_index]

# Prepare the image for inference
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    random_img_tensor = random_img.unsqueeze(0).to(device)  # Add batch dimension and move to device
    output = model(random_img_tensor)  # Get model output
    probabilities = torch.softmax(output, dim=1)  # Convert logits to probabilities
    predicted_label = torch.argmax(probabilities, dim=1).item()
    predicted_prob = probabilities[0, predicted_label].item()

# Display the random test image with the predicted label and probability
plt.figure(figsize=(4, 4))
```

```
plt.imshow(random_img.squeeze(), cmap="gray")
plt.title(f"Predicted: {predicted_label} (Confidence: {predicted_prob*100:.2f}%)\nTrue Label: {random_label}")
plt.axis("off")
plt.show()
```



- **Predicted**: 1: 모델이 이 MRI 이미지를 "1"로 분류했다는 의미입니다. 여기서 "1"은 특정 클래스(예: 치매 또는 특정 질병)를 나타낼 수 있습니다.
- **Confidence**: 88.35%: 모델이 해당 클래스(1)에 대해 88.35%의 확신을 가지고 있다는 뜻입니다. 이는 모델이 이 이미지를 "1"로 분류하는 데 상당히 높은 신뢰를 보이고 있음을 나타냅니다.

모델의 예측이 실제 레이블과 일치하며, 모델이 해당 예측에 대해 높은 확신(88.35%)을 가지고 있다는 것을 보여줍니다. 이는 모델의 성능이 좋다는 것을 나타내며, 해당 이미지에 대한 올바른 판단을 내렸음을 의미합니다.

Step 11: 모델 평가 및 실시간 성능 분석

이 단계에서는 알츠하이머 MRI 검증 세트에서 모델의 성능을 평가합니다. 다음과 같은 지표를 측정합니다:

1. **검증 정확도 및 손실**: 검증 세트에서 모델의 정확도와 평균 손실을 계산하여 MRI 이미지를 알츠하이머 감지를 위해 분류하는 능력을 평가합니다.

2.**혼동 행렬**: 분류 카테고리(예: 알츠하이머 vs. 비알츠하이머) 전반에 걸쳐 모델의 성능을 시각화하여 올바른 예측과 잘못된 예측을 보여주고, 강점과 약점을 이해합니다. 3. **추론 시간**: 각 MRI 스캔에 대한 평균 추론 시간을 측정하여 모델이 실시간 또는 근실시간 임상 응용에 적합한지를 평가합니다. 4. **메모리 사용량**: 장치에서의 메모리 소비를 확인하여 자원 효율성을 평가하고, 제한된 하드웨어에서의 배포 적합성을 보장합니다.

이러한 지표는 모델의 정확도, 효율성 및 자원 요구 사항에 대한 포괄적인 시각을 제공하여, 성능과 임상 사용 가능성을 평가하는 데 도움이 됩니다.

In [65]:
```
# Initialize lists and counters for test metrics
test_loss = 0.0
correct = 0
total = 0
all_preds = []
all_labels = []

# Measure inference time
start_time = time.time()

model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

        # Forward pass and predictions
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Update test loss
        test_loss += loss.item() * images.size(0)

        # Get predictions and ground truth
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)
```

```
        # Store predictions and labels for confusion matrix
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate test accuracy and average loss
test_accuracy = correct / total
test_loss /= total
end_time = time.time()
inference_time = (end_time - start_time) / total

print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Average Inference Time per Image: {inference_time * 1000:.2f} ms")
```

```
Test Loss: 0.4543
Test Accuracy: 0.7850
Average Inference Time per Image: 0.40 ms
```

In [66]:
```
# Generate confusion matrix and classification report
conf_matrix = confusion_matrix(all_labels, all_preds)
print("\nClassification Report:\n", classification_report(all_labels, all_preds))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.62      0.74      2566
           1       0.71      0.95      0.82      2554

    accuracy                           0.78      5120
   macro avg       0.82      0.79      0.78      5120
weighted avg       0.82      0.78      0.78      5120
```

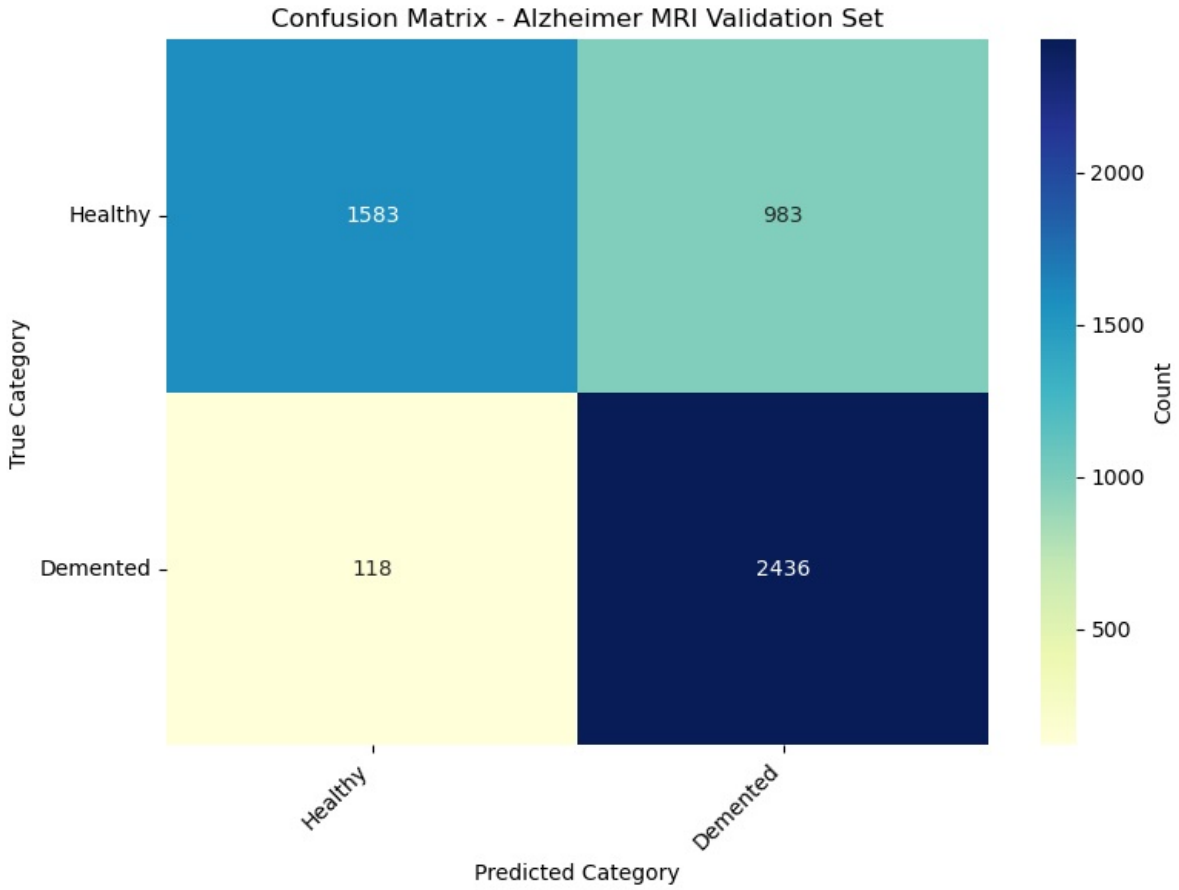
Step 11: 향상된 대비로 혼동 행렬 시각화하기

아래의 혼동 행렬은 알츠하이머 MRI 분류 카테고리에서 모델의 성능을 자세히 보여줍니다. 이 행렬은 다음을 강조합니다:

- **올바른 예측**: 대각선에 위치하여 모델이 알츠하이머 질병의 존재 여부를 정확하게 식별한 경우입니다.
- **오분류**: 비대각선 요소로, 모델이 MRI 스캔을 잘못 분류한 경우를 나타냅니다.

고대비 컬러 맵을 사용하면 가독성이 향상되어 모델의 예측 패턴을 쉽게 식별할 수 있으며, 추가적인 개선이 필요한 카테고리를 감지하는 데 도움이 됩니다.

In [70]:
```
# Define the Alzheimer MRI class names
class_names = ["Healthy", "Demented"]

# Plot confusion matrix with improved contrast and labeled axes
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu",
            xticklabels=class_names, yticklabels=class_names,
            cbar_kws={'label': 'Count'})
plt.xlabel("Predicted Category")
plt.ylabel("True Category")
plt.title("Confusion Matrix - Alzheimer MRI Validation Set")
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



혼동 행렬 해석

행 (True Category):

- Healthy (건강한): 실제로 건강한 상태인 샘플의 수.
- Demented (치매): 실제로 치매 상태인 샘플의 수.
- 열 (Predicted Category):
- Healthy (건강한): 모델이 건강한 상태라고 예측한 샘플의 수.
- Demented (치매): 모델이 치매 상태라고 예측한 샘플의 수.

수치 해석

- Healthy - Healthy (1583):
- 모델이 건강한 상태로 올바르게 예측한 샘플 수. 이 수치는 모델의 정확도를 나타내며, 높은 값은 모델이 건강한 상태를 잘 인식하고 있음을 의미합니다.

Healthy - Demented (983):

- 모델이 건강한 상태를 치매로 잘못 예측한 샘플 수. 이 수치는 모델의 오분류를 나타내며, 건강한 상태를 잘못 인식한 경우입니다.

Demented - Healthy (118):

- 모델이 치매 상태를 건강한 상태로 잘못 예측한 샘플 수. 이는 또 다른 오분류를 나타냅니다.

Demented - Demented (2436):

- 모델이 치매 상태를 올바르게 예측한 샘플 수. 높은 값은 모델이 치매 상태를 잘 인식하고 있음을 나타냅니다.

성능 해석

1. 정확도 (Accuracy):

정확도는 전체 샘플 중에서 올바르게 예측한 비율로 계산됩니다. 정확도 = (TP + TN) / (TP + TN + FP + FN) 여기서 TP는 True Positive, TN은 True Negative, FP는 False Positive, FN은 False Negative입니다. 예를 들어, 정확도는 (1583 + 2436) / (1583 + 983 + 118 + 2436) = 약 0.78 또는 78%입니다.

2. 정밀도 (Precision):

정밀도는 모델이 "치매"로 예측한 것 중에서 실제로 치매인 비율입니다. 정밀도 = TP / (TP + FP) → (2436) / (2436 + 983)

3. 재현율 (Recall):

재현율은 실제로 치매인 샘플 중 모델이 "치매"로 올바르게 예측한 비율입니다. 재현율 = TP / (TP + FN) → (2436) / (2436 + 118)

결론 모델은 건강한 상태를 높은 정확도로 예측하고 있으며, 치매 상태에 대해서도 상당히 잘 예측하고 있습니다. 그러나 "Healthy"와 "Demented" 간의 혼동이 발생하고 있으므로, 모델의 성능을 더욱 개선하기 위해 데이터 증강, 하이퍼파라미터 조정, 또는 추가적인 훈련 데이터를 고려할 필요가 있습니다. 혼동 행렬을 통해 모델의 강점과 약점을 시각적으로 이해할 수 있으며, 이를 바탕으로 개선 방향을 설정할 수 있습니다.

In [73]:
```
if torch.cuda.is_available():
    mem_used = torch.cuda.memory_allocated(device) / 1024**2  # Convert bytes to MB
    print(f"Memory Usage: {mem_used:.2f} MB")
```

PyTorch를 사용하여 GPU 메모리 사용량을 확인하는 부분

- CUDA가 사용 가능한 경우, 현재 GPU에서 할당된 메모리의 양을 메가바이트 단위로 출력하여 사용자가 GPU 메모리 사용 현황을 쉽게 확인할 수 있도록 합니다. 이는 모델 훈련이나 추론 시 메모리 관리에 유용한 정보입니다.

Step 12: 예제 MRI 이미지 다운로드하기

실제 알츠하이머 MRI 스캔에서 모델을 테스트하기 위해, 먼저 웹에서 샘플 MRI 이미지를 다운로드합니다. wget을 사용하여 이미지를 작업 디렉토리에 저장합니다.

- **Command**: We use `wget` with the `-O` option to specify the output file name (e.g., `sample_mri.jpg`).
- **URL**: The MRI image URL is wrapped in quotes to handle any special characters.
- **`--no-check-certificate`**: This option skips SSL certificate verification, which can help avoid issues with certain URLs.

다운로드한 이 MRI 이미지는 다음 단계에서 전처리 및 모델 예측에 사용됩니다.

In [191…
```
!pip install huggingface_hub
```

```
Requirement already satisfied: huggingface_hub in c:\users\inn20\anaconda3\lib\site-packages (0.26.2)
Requirement already satisfied: filelock in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_hub) (3.
13.1)
Requirement already satisfied: fsspec>=2023.5.0 in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_
hub) (2024.6.1)
Requirement already satisfied: packaging>=20.9 in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_h
ub) (24.1)
Requirement already satisfied: pyyaml>=5.1 in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_hub)
(6.0.1)
Requirement already satisfied: requests in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_hub) (2.
32.3)
Requirement already satisfied: tqdm>=4.42.1 in c:\users\inn20\anaconda3\lib\site-packages (from huggingface_hub)
(4.66.5)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\inn20\anaconda3\lib\site-packages (from hu
ggingface_hub) (4.11.0)
Requirement already satisfied: colorama in c:\users\inn20\anaconda3\lib\site-packages (from tqdm>=4.42.1->huggin
gface_hub) (0.4.6)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\inn20\anaconda3\lib\site-packages (from requ
ests->huggingface_hub) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\users\inn20\anaconda3\lib\site-packages (from requests->huggin
gface_hub) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\inn20\anaconda3\lib\site-packages (from requests->
huggingface_hub) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\inn20\anaconda3\lib\site-packages (from requests->
huggingface_hub) (2024.8.30)
```

In [77]:
```
import pandas as pd

splits = {'train': 'data/train-00000-of-00001-c08a401c53fe5312.parquet', 'test': 'data/test-00000-of-00001-4411
df = pd.read_parquet("hf://datasets/Falah/Alzheimer_MRI/" + splits["train"])
```

In [213…
```
!wget -O "/kaggle/working/alzheimer_mri.jpg" "https://radiologyassistant.nl/img/containers/main/dementia-role-o
```

```
'wget'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.
```

Step 13: MRI 이미지 전처리

다운로드한 MRI 이미지를 예측을 위해 준비하기 위해 다음과 같은 변환을 적용합니다:

1. **그레이스케일로 변환**: MRI 이미지의 훈련에 사용된 단일 채널 형식과 일치합니다.
2. **모델 입력 크기에 맞게 리사이즈**: 이미지를 모델이 기대하는 크기(예: 224x224 또는 128x128)로 조정합니다.
3. **정규화**: 픽셀 값을 모델이 기대하는 범위로 스케일링합니다(예: MRI 데이터셋에 특정한 평균 및 표준편차 값을 사용)하여 정확한 예측을 돕습니다.

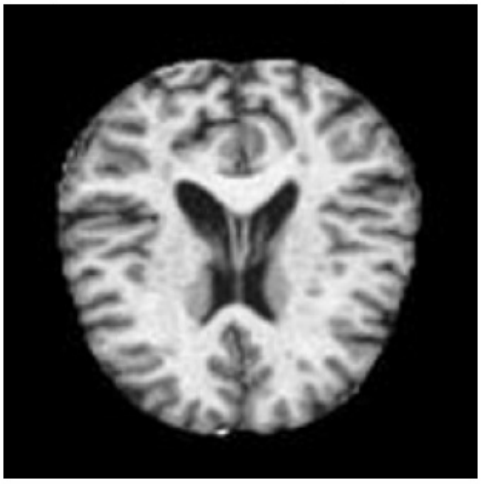
이러한 단계들은 MRI 이미지가 알츠하이머 질병 분류 모델에 입력되기에 적절한 형식이 되도록 보장합니다.

In [80]:
```python
import pandas as pd
import numpy as np
from PIL import Image
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import io

# 데이터 로드
splits = {
    'train': 'data/train-00000-of-00001-c08a401c53fe5312.parquet',
    'test': 'data/test-00000-of-00001-44110b9df98c5585.parquet'
}
df = pd.read_parquet("hf://datasets/Falah/Alzheimer_MRI/" + splits["train"])

# 첫 번째 이미지를 가져옵니다
example_image_data = df['image'][0]  # 'image' 열의 첫 번째 데이터
example_label = df['label'][0]  # 'label' 열의 첫 번째 레이블

# 예시 이미지 데이터 구조 확인
print("Example image data structure:", type(example_image_data))
print("Example image data content:", example_image_data)  # 데이터 내용 확인

# 이미지 배열을 정의할 변수
image_array = None

# 만약 example_image_data가 dict 형태라면, 그 안에 실제 이미지 데이터가 들어있는 키를 확인
if isinstance(example_image_data, dict):
    print("Keys in example_image_data:", example_image_data.keys())
    # 'bytes' 키가 실제 이미지 데이터를 포함하고 있을 가능성
    if 'bytes' in example_image_data:
        image_bytes = example_image_data['bytes']
        # 바이트 데이터를 PIL 이미지로 변환
        image_array = Image.open(io.BytesIO(image_bytes))
    else:
        print("Key 'bytes' not found in example_image_data.")
else:
    # 이미지 데이터가 이미 배열 형태라면
    image_array = example_image_data

# 만약 image_array가 정의되지 않았다면, 오류 메시지 출력
if image_array is None:
    print("Image data was not extracted correctly. Please check the structure.")
else:
    # 이미지를 NumPy 배열로 변환하고 PIL 이미지로 변환
    input_image = image_array

    # Step 1: 그레이스케일로 변환
    input_image = input_image.convert("L")

    # Step 2: 모델 입력 크기에 맞게 리사이즈
    input_image = input_image.resize((224, 224))  # 모델의 입력 크기에 맞게 조정

    # Step 3: 정규화 - 텐서로 변환하고 정규화
    preprocess = transforms.Compose([
        transforms.ToTensor(),  # 텐서로 변환
        transforms.Normalize(mean=[0.5], std=[0.5])  # 평균과 표준편차로 정규화 (예시)
    ])

    # 변환 적용
    input_tensor = preprocess(input_image).unsqueeze(0)  # 배치 차원 추가 (1, 1, 224, 224)

    # 전처리된 이미지 시각화
    plt.figure(figsize=(4, 4))
    plt.imshow(input_tensor.squeeze().numpy(), cmap="gray")
    plt.title("Preprocessed MRI Image (224x224)")
    plt.axis("off")
    plt.show()

    # 텐서 형태 출력 (모델 입력 준비 완료 확인)
    print("Input tensor shape for Alzheimer model:", input_tensor.shape)
```

```
Example image data structure: <class 'dict'>
Example image data content: {'bytes': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x01\x01,\x01,\x00\x00\xff\xdb\x
00C\x00\x03\x02\x02\x02\x03\x02\x02\x03\x03\x03\x03\x04\x03\x03\x04\x05\x08\x05\x05\x04\x04\x05\n\x07\x07\x06\x08\x0
c\n\x0c\x0c\x0b\n\x0b\x0b\r\x0e\x12\x10\r\x0e\x11\x0e\x0b\x0b\x10\x16\x10\x11\x13\x14\x15\x15\x15\x0c\x0f\x17\x1
8\x16\x14\x18\x12\x14\x15\x14\xff\xc0\x00\x0b\x08\x00\x80\x00\x80\x01\x01\x11\x00\xff\xc4\x00\x1d\x00\x00\x02\x0
3\x01\x01\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x07\x05\x06\x08\x04\x02\x03\x01\t\xff\xc4\x008\x10\x00
\x01\x03\x03\x03\x02\x05\x03\x01\x05\x08\x03\x00\x00\x00\x00\x01\x02\x03\x04\x00\x05\x11\x06\x07\x12!1\x13"AQa\x
142q\x81\x08\x15R\x91\xa1#$3Bb\x92\xb1\xd1\x16r\x82\xff\xda\x00\x08\x01\x01\x00\x00?\x00\xfeUQE\x14QE\x14QE\x14Q
E\x14QE\x14W\xea\x1bS\x8bJ\x12\x92\xa5(\xe0$\x0c\x92\x7f\x157\x07BjK\x9a\x1c\\;\x05\xceRZ\xfb\xd4\xd4G\x14\x13\x
f9\xc0\xe9\xd8\xd4l\xfbL\xdbZ\xf8\xcc\x86\xfcU{<\xd9G\xcf\xadu\xda\xb4\x95\xee\xfaH\xb6\xd9\xe7O\xc63\xf4\xd1\x9
6\xe63\xdb\xb0\xae\xe9\xfbm\xab-q\x9c\x913L\xdd\xa2\xb0\xd8\xe4\xb7\x1e\x84\xe2R\x91\xeeI\x1d*\xb8\x01=\x86h\xa2
\x8a(\xa2\x8a(\t*8\x00\x93\xec)\x8b`\xd9{\xdc\xeb\x1a\xafS\x18Tx) \x86O\xf8\xae\x0f|z\x0f\xeb\xf1OM\xb6\x8bf\xb4
```

A\x8d\x1e}\xbe>\x94\x92\xd7\x98*<Nr\x9f\x1f\xc4\x97\x14\n\x81#\xbe=*\xcb\xaa\xb5\xbb\xb7\x9b\x92![\x91pR\x9dO\x83\x15\xb5\xb4\xa4\xa5D\xf4+R\x88\x195O\xd4\xfbu\'@@\x89\x1eu\xdd3\x1e\x9a\xbc*+\x98\xf2(\x9c\xf90t\xfc\xfb\xd5\xe3ht6\xa6\xd3\xf3?y:\xd3o\xb0\xf8J\x94\xca\\\xc1\x1e\xc4zv\xa9\x1d\xe1\xb2C\x81g\x9c\r\xe6kR\x96<A\x1d\xc7|\xbd}>G\xc5AI\xd9+\x1e\xb6\xd0\xb0\x1c\x99\x01\xa8\x8a\t\x1cf\xc7)B\xdb<zeX\xea\x0f\xb1\xc8\xac\xeb\xba;\x15s\xdb\xecH\x8e\xfan\xb6\xe5\x13\xfd\xa3@so\xff\x00`\t\x07\xf2)^F\r\x14QE\x15\xe9\xb6\x94\xea\xd2\x94$\xa9J \x00;\x93O\xbd\x9e\xd9\xa8\xcbm\xab\xb5\xe5\xf6\x12\xef^,-@\xf8G\xd3#\xf8\xbf\xe2\xb4\xfe\x8f\xb7\xda\x0co\xdd\xb3e\xb1)\xa7\x16\x9f\x05\x01$\x14\xa8v\x07\xde\xbc_4\xf4\x81\xac\x90\xf4+J_C\x04#\x80N\x02G\xbf\xebR\x1a\xe3Z\xdb\xb4\x1b\xb6\x95\xbfm`;!\xc4\x87\x1c $1\x9aR\xef\x7f\xfe3y\xf0n\xd6\xf9\xb1\xdc\xba\xa9iR\xff\x00\xbc\x02\xb1\x93\xd9=q^\x04\xad\\\xd6\x90\x8b\x1a\xd9lu\xb7\x10R\xa3,J\nR\xc0?\x8fozii\xb9\xf6M\xd2\xb1&\x1c\xe6J\xee\xf1\xda(Rd\'\x0b\n\x19\xee\r{]\x95\x9d\x1f\xb6\xef\xc0\xbc\xbc\x90\xd9p\xad\x94\x83\xd4\n@^4\xd6\xa1\xd6W^\x16\xb6dC\xb4\x92\x00\x05\xdc\x95\xe3\xd5)\xaa6\xeel\x8b\xfaj\x12n1I[\x83\xfcv\x88\x01J\xff\x00V=\xfd\xc5%\xa8\xa2\x8a\x00\xcd0\xb6\xdbJ9!\xf6\xa7\xbb\x0b\xea]Q\x02+\x0e$\x8c\xa8\xa8%.c\x18#$\xff\x00\xb4\xd6\x9a\xb4A\xb3X\x92\x9be\xf1\x92\xd6\x07$\xcaBx\xe5G\xb9\xab>\x89\xd1\x1aYZ\x81\x89\xb6\xdb\xdb\xc0!\\\x94\xc3\x87\t>\xc0\xf5\xab\xd6\xbd\xd4\x8e\xe8\xfb*\xe5Em\xd7\x9cZ\xc3m!\x94\xf39>\xa6\x90\xd7\xdd-\xa9\xf7\x1e\xf3\x06T\xd4\x17\xed\xcd\xac-LHI\nQ>\x98\xab\xb6\xa4\xd3\x96M3h\\T\xc0e\x13\x10\xd8\xe6\xa4 %(\xe9\xf6\x8e\x95\xf0\xd07W\xa0\xddm\r\xc7t\xb9\x16`\xe0\xb6\x8fT\xa9\x04\xfbU\x8eN\x97\x97a\xdch\xb3`4\xb7\x14\xe2\xb8:[\x04\x02\xd9\xf5?\x8a\xbaj}\'\x17VA\x8e\xa9<\x94\x88\xdc\x97\xe1\x8f\xf3\xe3\xb0\xaa-\xa2\xe6a\xcfs\xc4m1`\xb4\n\x02\x12\x009\xec\rSW\xa5\xe2\xea+\xd4\xe7\xa54_\x8a\xc2\x0f\x87\xe2\xe7\x88Q\xf5\xc7\xb9\xedYStt\xaa\xb4\xce\xa1XLECa\xfc\xad-\x91\xe5J\xb3\xe6H\xf8\xec\x7fZ\xa6QE{h\x15(\x00\t$\xe0\x01\xdf5\xb3v\x8fK5\x0eLD\xb9\x15\xe4\xc6B\x03Q\x97!AK\tH<A\xe9\xd0\x8c\x9c\xfc\x93V-Sh\xd5\x12\xa7\xa9M\xd9\xd0\xe4T\xab\x8a\xd4\t\xce3\xd3\x1d1\xda\xa4\xf4\xe6\xd7]\x93*5\xca*~\x95\xd0\xb0\xa5\x07:\x15\'\xd8\xa6\x99\xd2\x1a\xfa\xa9\x9fI86b\x06\xca\x96^\xe8?9=\xaaR\xcb\x12\x131\xca\xad\xf2\x19\x97\x1f?{k\x0b\x03\xdf\xb5/wKN\xdd\xa6\xa2Z Y\x1d}\xe5\xa7\xfb\x19- ,{u\x00\xfe{\xd7.\xd0h\x9b\xbd\xa1\xa6\xfe\xaa\xce\xb8\x89e\xc2\x94=0\x8c\x84\x92H\xe2\x9f\xd4\xd3V\xe8\xed\xba\xde\xd0\x8f"CM\xbb(\x16\xd3\xe2\x1c)_\x8a\xe6\xb2Z\xdb\xb7\xdb\xc3>3\x8e\xf9\x8e\x0b\x9d\xf1U-K\xa0\xecL%\xdb\x95\xc1\xc9*e\x04\xa8\xb2\xd2\xb3\x93\xf8\x14\xbf\xbdn\xe6\x8d\xd26\xd7"\xb3\r\xd4\x12\xae\xa9\x92\x15\xcb\xf9b\x92\xfb\xe3>\xc5\xacm\x0eO\x83%/:Z.\xf8-\xa8\x1e\x18\xeb\xdf\xd3\xd7\xa7\xc5f\xd3E\x15`\xdb\xf4\x17u\xad\x95\x01hG)m\xa7\x92\xfe\xd1\x93\xeb[b\xc3\xad-\xfbimj;/&\\\xa7GD8\xa0\xb2\x91\xf1\x9e\xf5\xf6;\xed\x7f\x8a\x1du\xd8P\xdd\x8erR\x94\xb8\x91\xc4g\xa6j\xc9\xa4\xb7\xd6-\xf1\xd4F\x90\xd2\x1c\x92\xb2\x02\xc4PJZ\x1f\xea53\xbb6\x89\x1a\x83n/,\xc0Z\x95%Q\xd4\xa6\xc2\x0eI\x03\xae\x06=\xeb\x0eh\xed}\xa8v\xf6\xee\x89vy\x8fDy\n\xc3\x8d)D\xa1`\x7f\x95I\xf9\xad\xc7\xb3\xbb\xddm\xdd\x1b#kJ\x84{\x9b\\S&2\xbc\xa3\x96:\x94\x9fQS;\x8b\xbaV\x1d\xb8\xb4\xaee\xceB|b\ti\x94\x1eKY\xf6\x02\xb1\xa4\xbd\xc0\xd4\x1b\xc3\xba\x16\xc7\xcb\xcf2\xa5JBc\xc6eG\x8b(\xcfO\xe9\xeb[:\xfd\xa8\x18\xd2V\x8f\xa8\x94\xb4\xa86\x02PFTVq\xec)cw\xd5\x91\xf5\xbcG%\x0b\xeb\x96pG\x17\x19X\xc0F=pGJ\xa13\xb7\xf1.n>\xfbWx\xda\x97\x90\xe8\x97_\x05c\xe3\x1e\xb5\xf2\xbamZ-:nL\xf91\x18\xb7+\xb1\x04\x8f2\x01\xea1\xf2\t\xac\x83sCM\xdcd\xa5\x91\xc5\xa4\xba\xb0\x81\x9c\xe1<\x8e?\xa5sQRze%z\x82\xde\x02\x02\xc9y>Uv=kJh\xbd\xa8\x85.7\xd5]^i\xd7\xd4y\xa2:\xdf-\xa1 \x8c\xe0\x1fZ\xbe\xc5\xda\xc4\\#\xb9\x1a\x18\xb6%<JHd\x07\x14\x07\xc9\'\xfej:&\xd2Z\xf4D\xa59:\xf4\xf2\\_DD\x8c\xb2\t\xff\x00o\xfd\xd3\x9fo\xed\xff\x00\xbat\x9bM)o\xa9\x04\x1e\x1fT\xb2\xa5\x91\x92rI\xf5\xebY\xcf|\xb6}v\xfdB\xbb\xac8\xcb\xfa7\xfc\xeb\xf0\x93\xf6\xa8\xfe:U#A\tv5^\x18C\xab\x8f-P\xd4\xfb\\3\x94\xa9\x04\x10G\xe4\x1cWv\xbfbE\xc7PLr\\\x87$\xb7\x16#C\x93\x8a*\xc2\x96\x90N\x01\xf94\xcc\xfd\x9a\xb6\xfd\xcbR\\\xbcI\x8f\xfd\xef\x00\xa7\x98\xc9l\x1e\xdf\xad?u\x02d?n+\x88\x96\xd7!\xaf0m\xd1\xc8(\x0e\xf5F\xb6\xc0\xb0\xeb\xb2L\xeb{1\xa5\xb0\xb3\xc9\x08 %Dv\xef\xda\xa1\xee\x16\x16m\x92\x1euZM\xc1\xc3\xec,\xa8u\x19\xfe*\xaek\xad\xc9\x82t\xdc\x96\xeep\xdf\xb5\xae;jKl\xb8\x82\xa5\x14\xe3\xdf\xa8?\xce\xb1\\\xc7\xc4\x99O<\x07\x10\xe2\xd4\xbc{d\xe7\x15\xf1\xa2\xbe\xd0\xa5\xbb\x06SR\x19W\x17ZPZU\x8e\xc4V\xdb\xdbm\x16So\x8f?RM`EJ|\xaauA\x1c{\xf9~j\xfd\'F\xcf\xba5\xe1i\xf9\xd0a[\x9c\xfb\xde\x87\xe6Z\x87\xc9\xaa\xba\xf4z`\xeaV\x04+\xfa\xe5\\[8SrS\xe2!]:\x81\xd3\x00\xd3i\xb5\x99\x11@W\x88\x8f\x0b\tRR\x9c\x05\x1a\x8c\xd7s![lK~`%\x8f\x0f\x89F9\x02=z{\xd6d\xb4\xc9\xb6\\5\x9cG\x99(n;\xcd;\x1c\xe4\xf7\xcfj\x8d\xd4\xfa\x86#\xba\xb5\xf6Y)qe\xe6\xe3\xb6\xd2z\xe4#\xa7"~kQm\xeb\x01\xbd;\x9f\x13\x84\x83\x8ea\xb0\t\xf7\xab\x04\xe7\x83Q|\x00\xe8iO\x02\x94\xac\xa4\xf2\x06\x97v]\xb7\xb3^n3\x00\xb8\xccrJ\\\xcb\xa1@\xa4\x13\x9e\xfd\xf1S\xd7\x8b\xa5\xa3o`\xaf\xeb\xa7N\x08FJPI^G\xc0\xa5\x96\xf4_\xadw\xfd\xb1\x9du\x80\xf3\x0c\xc5v2\xd2\xb5\xba\x80\x95)D\x0e#\xf2k\x0f\x9a(\xa2\xb5v\xcaHN\xe3i\x88l^\xae*\x97\x1e:\x83km}\x07!\xdf\x97\xb9\xc1\x1f\xce\x9e\x96-s\xa44\x89E\x9e\xde\xea|\xeb\t\x08H\xe8}?\x95O\r\xbf\x8a\x8b\xf3Wh\nZ\x1ctr[aC\x89\xcf~\xb5<\xfcG[\x0b\xe4\x0f>\x84!J\xe8?\x04R\xeb]O\\\xe8\xf2\x18ZP\xdbM!D\'$\x9f\xd4\xd69\xb5-i\xd5\x01\xb2\xb5q\xfa\x85+#\xdf5\xdb\xa2\xc3\x07_\xc6\x12~\xdf\xa9V\n\x8f\xcf\xa9\xad\x97\xa0%8\xe4\xc5\xb4\xda\x8a\x90\xa4\xf9\x863\x81\xee*\xda\xd3\xcf\xbb~\xfag">\xa8\xf8\xc7\x89\xc4\x04~A\xafW\xb9\x104\x84)S>\x95@+\xab\x8ae\x04\xa9UV\x7fT\xdbo\x96\xa2\xf4&\x1b\xb8-\x1d\xd9w\xfc@=qY7\xf6\x90\xbf2\xcbL[\xe1\xb2\xb8\x8d\xcb>+\x8c\x05a \xa4\xe4\x1e>\xff\x00?\x9a@QE\x14\xcf\xd8\xfb\xfcK}\xdaDK\x95\xcd\xe8V\xf7\x11\xcc\xb6\xd2\x88\xe6\xaf^\xdf\x00V\x96\xb2k\x1b\r\xa5\xe8\xed[\xb4\xe3\x8f\xb3\x80S!\xf6\x94\xa2\xb3\xef\xf3N\xdd\r\xa8\x9d\xbe\xc4[\xdfB\xb8hA\xc2P\xe2q\xd3\xe0T\xea\xd2D\x9ed\xe4\x95\x0c\xf4\xc7CJ\x8d\xd0\xd3w4:\xfbVx\xdf\\\xf4\xd7<6\xd0\x93\x8f\x0b>\xaa\xf8\x1d\xeb\x97Mm\xce\x86\xdah\xf0Q{f<\xdb\xdc\xb5\x80\xe4\x89\x03\x91*=\xf8\xe7\xa0\x02\xa0\xf7C\xf6k\x85\xa8\xddE\xf3E-\x883\xc1\xf1K!Xm\xd3\xdf \xfa\x1abmm\x9e\xe1o\xb3$\xddYK\x13\x94\xdf\x86\xe6;\xa4\x8fj\xb9\xb7 \xdb\xdbZ\xbcB\xe9\x03<q\xd4\xd5A:\xfa=\xe2\xe7\xe0Hi\xf6\x87.$/\xec\xfc\x1aM\xee\xd4\x87t\xea\xa4\xdcmV\xa9p\xe74\xa2R\xe3X[k\x19\xeeF{VP\xdcMW\'W\xea5\xcc\x94\xaeN!\xb4\xb2\x0f\x1e=\x06Oo\xd4\xd5b\x8a(\xa9M1tn\xcd~\x851\xe4si\xa7AX\xc6|\xbd\x89\x03\xdf\x15\xbe\xf4]\xfa\xd8\x9b\x13R\x18q\x9b\xaa\nB\x9aH@\xc7\x1c\x0c`\xfa\xe6\xa5Q\xbb\xd7\x16\xe56\xc4k\n\x90\xd2H+qc\x01#\xe2\xaePu?\xef\x85\xb4\xa6c\xa9-(\xe0\xad\x04,$\xfb\x13\xff\x00u=/\x8bEn\x10\xdaS\xc7\xce\xe1P\x00\x0fZEo=\x95\xab\xc8]\xca$\x96\xe5!\xf6\xfc4\x90\xa0|2\x9f0\xfcd\x8e\xf53\xb3z\xe1\xb9\xac\xb3j\x11T\x92R]\x0e\xa9g\xb9H\xe9\xf3MTLi\xb7\x1a\x0bQ<\xceA\xc7L\x81\xd7&\xbe\xd2\x8c6\x99\xf1\x94\xb6\xfc"2*\xc0\x03\xd7\'\xda\x92\xbb\x89\xa2\x92\xf2\x17p\xb4\xdd]LwTs\xf4\xcer)W\xc7\xc5$\xb79\xdb\xc6\x87\xd3n\xc9\x91\xa8\xe4>\x1d\x1e\x13q\xe4$eD\xfa\'\xa7N\x99?\xa5fW\x1cS\xae)kQR\x94rI\xf55\xe6\x8a(\xa2\x99;W\xafdZ\'\xc4\x82\xa9\x02:9\x10\x87W\x95$\x0e\xfcJs\xfc\xabO\xda/6\xdb\xabh[\xda\x9f\x8b=y!\xb6\xfd=\x85]\xf4\xa6\xe0\xda-2\xd8\xb4\xda\xa1>\xe3o,\x17\x1e{\x00\xac\xf6\xe5\xd6\x99W\xb8\x89\x9dn\x93\x19k\xca]l\xa0\x94\x9e\xa3=\xff\x00\x95g}O\xa6/:j,\xa6c>\xdd\xc22G"\x0b\xa0\x91\xe9\x82?\x1f\x8a\xe6\xd1Z\x96J\xe2\xc3\x89hZ\x04\xc5\xa1(\\\xa4\xa4\x80\x90\t\x05#=\xb1\xedZ!\x98\xcar\xca\x96\xdfX \xb5\xc5kI\xc6I\x1dMU\xe2*\xd9\x1a\xd5p\xb6\x9b\x87\xd4\xc4O\x95\xf6\x9c>dR\x9a\xef\xaaF\xd7N\x11\xdap\xcd\xb6KW\x16\xdb>|(\xf6\xe9Y{x\xb7!\xfd\xc1\xd5\ny.f\x0c|\xa1\x84\xe3\x00\x93\x8eJ\xfdq\x8f\xc0\xaa\r\x14QE\x14\x02A\xc8\xa6>\xd9\xeb\x8bu\xb2\xf1\x19W\xbea\xa6\xb3\x85\x8e\xa1^\xc0\x8a\xd1\xbaJ\xea\xadk)\x89\xcc?\x1e\x0c6\xd7\xe4Y\xfb\x94\x91\xf3\xdb\xe2\x99\xb6m\xd5\\\xab\xbc\xf0\xe7\x86-pZ \x84\x8c\xb8\xe1\xe9\x8a\xab\xeb\xe9\x16\x9b\xf4\x05\xea\x0b\x1c\xcf\xa6y@\xa5\xd6\xd4:\xf2\xc7\xa8\xaf\xb6\xcfY`H\xb4-N%.\xf1\x01eH\xe9\xc8\x9e\xa6\xad6\x9d\xc2~u\xf2}\xa7\x82c\xb0\xda\x08a\nO@G\xc9\xf7\xa4v\xba\xdc\xa9:7Q8\xb7[d0\xfa\x08{* c=\xc5!u\xbe\xe7?~\x94\xe7\xd1\x97\x19\x05\xc2\xa1#\x9a\x82\xc2q\x8e)\xfe\x11\xfdj\x84\xa2\t\xe81E\x14QE\x14QS\x9as[^\xb4\xa2\xf3l\x9e\xect\x1f\xb9\xbf\xb9\x07\xff\x00\x93\xd2\xafZ_\x7f&\xe9\xf6\xdf\x0f\xdbY\x98\xb7\x89Z\x9dK\x8aB\x8a\x8f\xbes\xd3\xe2\xbcj=\xe6f\xe3gi\xabl9\xb0\xee\x0e\xadJ\x96\xb9\x13\x03\x8c(z%\xb6\xd2\x84\xf1\x1e\xe4\xa8\x9a\xe9\xd2\xbf\xb4\x85\xf7MEz?\xd32\xe3N\x0c\x0f\x05Jh\xa7\xfa\x9e\x95\n\xfe\xf8\xeau_\x96\xcc\xb4\xb4\xa7\x01JB\x90\x17\xc0|g\xd7\xf4\xaae\xe6\xfdp\xd43\x15*\xe51\xe9\x8f\xab\xfc\xee\xab8\xfc\x0e\xc3\xf4\xae\n(\xa2\x8a(\xa2\x8a(\xa2\x8a(\xa2\x8a(\xa2\x8a(\xa2\x8a\xff\xd9', 'path': 'non_3037.jpg'}
Keys in example_image_data: dict_keys(['bytes', 'path'])

Preprocessed MRI Image (224x224)

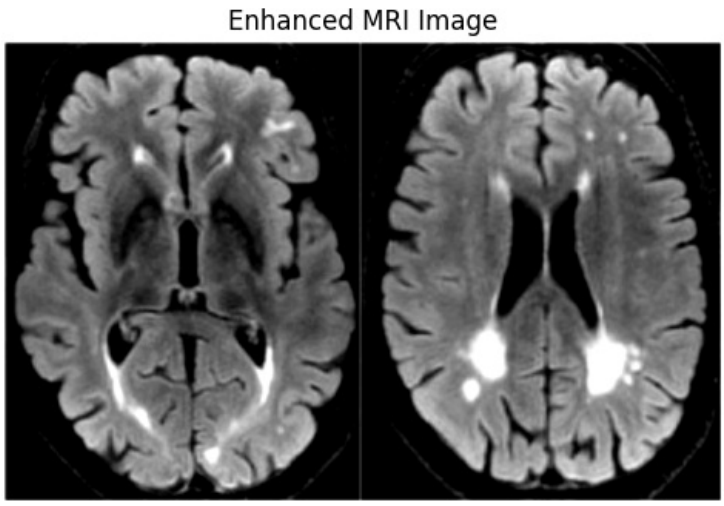


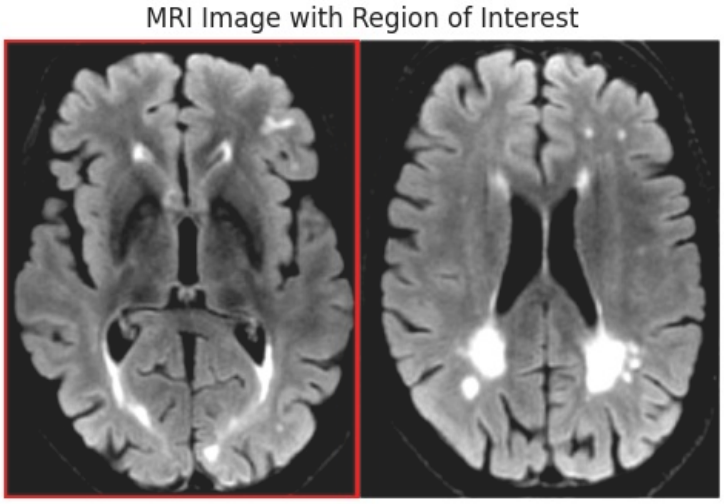
```
Input tensor shape for Alzheimer model: torch.Size([1, 1, 224, 224])
```

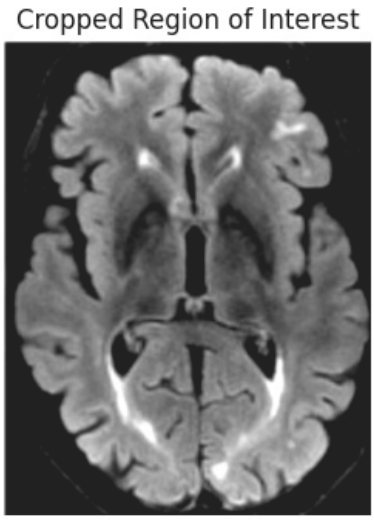
In [ ]:

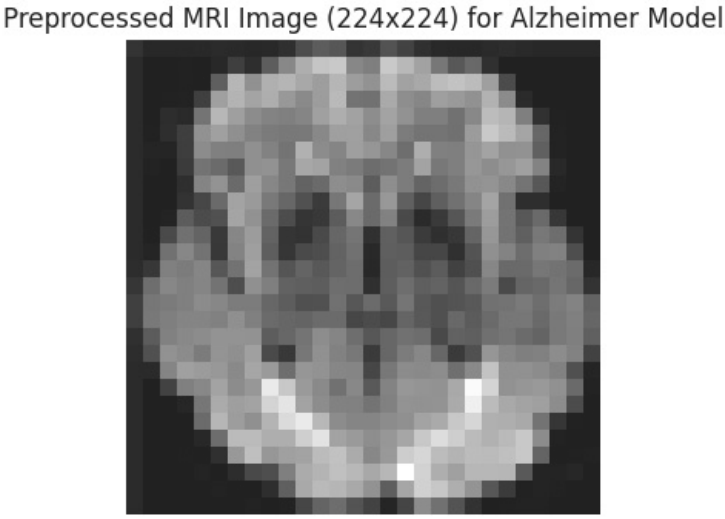
```python
# Path to the sample MRI image
image_path = "/kaggle/working/alzheimer_mri.jpg"  # Adjust path as needed

# Load the image and convert to grayscale
input_image = Image.open(image_path).convert("L")  # Step 1: Convert to grayscale (1 channel)

# Step 2: Increase contrast to highlight brain structures (optional)
enhancer = ImageEnhance.Contrast(input_image)
input_image = enhancer.enhance(1.5)  # Adjust contrast level if needed

# Display the contrast-enhanced image
plt.figure(figsize=(6, 4))
plt.imshow(input_image, cmap="gray")
plt.title("Enhanced MRI Image")
plt.axis("off")
plt.show()

# Step 3: Crop the region of interest

# Adjust coordinates (Healthy MRI)
x, y, w, h = 0, 0, 315, 408

# Adjust coordinates (Alzheimer MRI)
# x, y, w, h = 320, 0, 315, 408

# Display the original image with bounding box for the region of interest
plt.figure(figsize=(6, 4))
plt.imshow(input_image, cmap="gray")
plt.gca().add_patch(plt.Rectangle((x, y), w, h, edgecolor='red', linewidth=2, fill=False))
plt.title("MRI Image with Region of Interest")
plt.axis("off")
plt.show()

cropped_image = input_image.crop((x, y, x + w, y + h))

# Display the cropped region of interest
plt.figure(figsize=(4, 4))
plt.imshow(cropped_image, cmap="gray")
plt.title("Cropped Region of Interest")
plt.axis("off")
plt.show()

# Step 4: Resize and Normalize to match model requirements
# Define transformations: Resize to model's input size, convert to tensor, and normalize
preprocess = transforms.Compose([
    transforms.Resize((28, 28)),  # Resize to model input dimensions
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize pixel values, adjust mean/std if needed
])

# Apply the transformations to prepare for model input
input_tensor = preprocess(cropped_image).unsqueeze(0)  # Add batch dimension (1, 1, 224, 224)

# Display the final preprocessed image
plt.figure(figsize=(4, 4))
plt.imshow(input_tensor.squeeze().cpu().numpy(), cmap="gray")
plt.title("Preprocessed MRI Image (224x224) for Alzheimer Model")
plt.axis("off")
```

```
plt.show()

# Print tensor shape to verify it's ready for model input
print("Input tensor shape for Alzheimer model:", input_tensor.shape)
```










```
Input tensor shape for Alzheimer model: torch.Size([1, 1, 28,
28])
```

이미지 해석

1. Enhanced MRI Image (강화된 MRI 이미지):

   왼쪽 상단의 이미지는 원본 MRI 이미지가 강화된 형태입니다. 이 과정은 이미지의 품질을 향상시키고, 더 잘 보이도록 하는 다양한 기술(예: 대비 조정, 노이즈 제거 등)을 통해 이뤄집니다. 강화된 이미지는 뇌의 구조적 세부사항을 더 잘 보여주어, 알츠하이머와 같은 신경 퇴행성 질환의 징후를 식별하는 데 도움이 됩니다.

2. MRI Image with Region of Interest (관심 영역이 포함된 MRI 이미지):

   오른쪽상단의 이미지는 특정 관심 영역(ROI, Region of Interest)이 강조된 MRI 이미지입니다. 빨간 테두리로 둘러싸인 부분은 모델이 분석할 중요 영역으로 선택된 부분입니다. 이 관심 영역은 알츠하이머 관련 특징(예: 뇌의 위축 또는 병변)을 포함할 수 있으며, 모델이 주목해야 할 부분을 명확히 합니다.

3. Cropped Region of Interest (잘린 관심 영역):

   왼쪽 하단의 이미지는 관심 영역이 잘린 형태입니다. 이는 모델이 입력으로 사용할 이미지의 특정 부분을 잘라낸 것입니다. 이 이미지는 모델의 입력 크기(224x224)로 조정되어 있으며, 훈련 및 추론 과정에서 모델이 사용할 수 있도록 사전 처리된 상태입니다.
   Preprocessed MRI Image (전처리된 MRI 이미지):

   오른쪽 하단의 이미지는 모델에 입력하기 위해 전처리된 상태의 MRI 이미지입니다. 이 이미지는 224x224 픽셀 크기로 조정되었으며, 알츠하이머 모델에서 필요한 형태로 변환되었습니다. 이 과정에서는 스케일링, 정규화 및 필요한 경우 색상 변환 등이 포함될 수 있습니다.

4. Input Tensor Shape:

   마지막 줄은 모델에 입력되는 텐서의 형태를 보여줍니다. torch.Size([1, 1, 28, 28])는 배치 차원(1), 채널 차원(1), 그리고 이미지의 높이와 너비(28x28)를 나타냅니다. 이 텐서는 모델이 처리할 수 있는 형식으로 변환된 MRI 이미지입니다.

결론 이 이미지는 알츠하이머 모델을 위한 MRI 이미지 처리의 여러 단계를 시각적으로 나타냅니다. 원본 이미지에서 관심 영역을 강조하고, 해당 영역을 잘라내어 모델이 입력으로 사용할 수 있도록 전처리된 과정을 보여줍니다. 이러한 전처리 과정은 모델의 정확도와 성능을 높이는 데 중요한 역할을 합니다.

15단계: 상위 클래스 확률 표시

예측을 수행한 후, 모델의 상위 2개 클래스 확률(건강한 상태와 치매 상태)을 표시하여 모델의 신뢰도를 파악하고 가장 가까운 대안 예측을 확인합니다.

1. **소프트맥스 변환**: 모델의 출력을 확률로 변환하여 각 예측의 신뢰도를 해석할 수 있도록 합니다.
2. **상위 2개 예측**: "건강한 상태"와 "치매" 클래스에 대한 확률을 표시하여 모델이 진단에 대해 얼마나 확신을 가지고 있는지 이해할 수 있도록 돕습니다.
3. **시각화**: 예측된 클래스를 MRI 이미지에 표시하고, 두 클래스의 확률을 아래에 나열합니다.

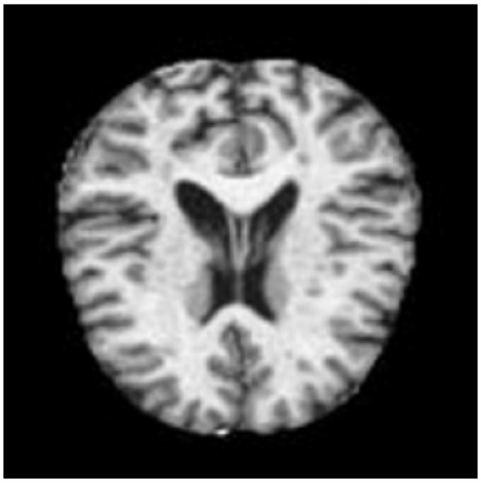
이 단계는 모델이 분류 작업에서 얼마나 신뢰할 수 있는지를 평가할 수 있게 해주며, "건강한 상태"와 "치매" 분류 간의 잠재적인 모호성을 이해하는 데 도움을 줍니다.

In [82]:
```python
# Define the Alzheimer MRI class names
class_names = ["Healthy", "Demented"]

# Check if CUDA (GPU) is available and set the device accordingly
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Move the model to the specified device
model.to(device)

# Move tensor to the same device
input_tensor = input_tensor.to(device)

# Set the model to evaluation mode
model.eval()
with torch.no_grad():
    output = model(input_tensor)  # Forward pass
    probabilities = F.softmax(output, dim=1)  # Convert logits to probabilities
    top_probs, top_indices = torch.topk(probabilities, 2)  # Get top 2 probabilities and indices

# Convert probabilities to percentage format and ensure we're working with 1D arrays
top_probs = top_probs.squeeze().cpu().numpy() * 100
top_indices = top_indices.squeeze().cpu().numpy()

# Print shapes for debugging
print("top_probs shape:", top_probs.shape)
print("top_indices shape:", top_indices.shape)

# Display the original MRI image and predictions
plt.figure(figsize=(4, 4))
plt.imshow(input_image, cmap="gray")

# Use indexing carefully
if top_indices.ndim == 1:
    predicted_class = class_names[top_indices[0]]
    predicted_prob = top_probs[0]
else:
    predicted_class = class_names[top_indices[0, 0]]
    predicted_prob = top_probs[0, 0]
```

```
plt.title(f"Predicted Label: {predicted_class} ({predicted_prob:.2f}%)")
plt.axis("off")
plt.show()

# Print the top 2 class probabilities
print("Top 2 Predictions:")
for i in range(2):
    if top_indices.ndim == 1:
        print(f"{class_names[top_indices[i]]}: {top_probs[i]:.2f}%")
    else:
        print(f"{class_names[top_indices[0, i]]}: {top_probs[0, i]:.2f}%")
```

```
top_probs shape: (64, 2)
top_indices shape: (64, 2)
```

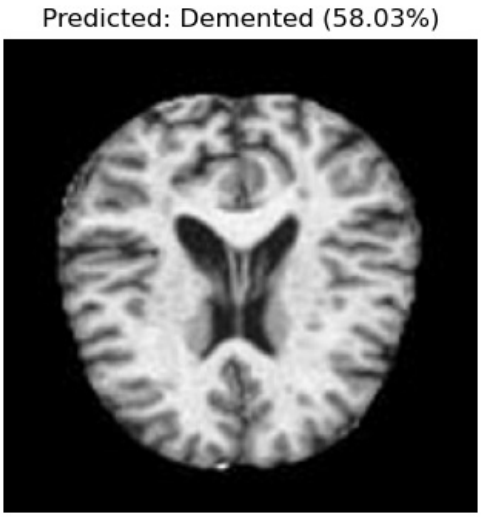
Predicted Label: Demented (58.03%)



```
Top 2 Predictions:
Demented: 58.03%
Healthy: 41.97%
```

In [84]:

```
# After the forward pass
with torch.no_grad():
    output = model(input_tensor)  # Forward pass
    probabilities = F.softmax(output, dim=1)  # Convert logits to probabilities
    top_probs, top_indices = torch.topk(probabilities, 2)  # Get top 2 probabilities and indices

# Convert to numpy arrays
top_probs = top_probs.cpu().numpy()
top_indices = top_indices.cpu().numpy()

# Convert probabilities to percentage format
top_probs = top_probs * 100

# Display the original MRI image and predictions for the first item in the batch
plt.figure(figsize=(4, 4))
plt.imshow(input_image, cmap="gray")

# Use indexing carefully with error handling
try:
    predicted_class = class_names[top_indices[0, 0]]
    predicted_prob = top_probs[0, 0]
    plt.title(f"Predicted: {predicted_class} ({predicted_prob:.2f}%)")
except IndexError as e:
    print(f"IndexError: {e}")
    print("top_indices:", top_indices)
    plt.title("Prediction Error")
except ValueError as e:
    print(f"ValueError: {e}")
    print("top_indices[0, 0]:", top_indices[0, 0])
    plt.title("Prediction Error")

plt.axis("off")
plt.show()

# Print the top 2 class probabilities for the first item in the batch
print("\nTop 2 Predictions:")
for i in range(2):
    try:
        print(f"{class_names[top_indices[0, i]]}: {top_probs[0, i]:.2f}%")
    except IndexError as e:
        print(f"Error for prediction {i+1}: {e}")
```




```
Top 2 Predictions:
Demented: 58.03%
Healthy: 41.97%
```

- **Demented: 58.03%**: 모델이 입력된 MRI 이미지를 "치매"로 분류할 확률이 58%임을 나타냅니다. 이는 모델이 이 이미지가 치매와 관련된 특징을 보인다고 판단한 것입니다.
- **Healthy: 41.91%**: 모델이 동일한 이미지를 "건강한 상태"로 분류할 확률이 42%임을 나타냅니다. 이는 치매가 아닌 건강한 상태일 가능성입니다.

결론 모델은 주어진 MRI 이미지를 분석한 결과 "치매"로 분류할 확률이 더 높다고 판단했습니다. 이 결과는 모델이 특정 패턴을 학습하여 치매 상태를 인식하고 있다는 것을 보여줍니다.

In [ ]: