```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import spacy
import numpy as np
from scipy.signal import find_peaks

# Load spaCy tokenizer
nlp = spacy.load("en_core_web_sm")

def load_text_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        text = f.read()
    sentences = [sent.text for sent in nlp(text).sents]  # Split text into sentences
    tokens = [token.text for token in nlp(text)]
    return sentences, tokens, text

class Encoder(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(Encoder, self).__init__()
        self.hidden_dim = hidden_dim
        self.bigru = nn.GRU(input_dim, hidden_dim, bidirectional=True, batch_first=True)

    def forward(self, x):
        h, _ = self.bigru(x)
        return h  # h ∈ R^(N × 2H)

class Decoder(nn.Module):
    def __init__(self, hidden_dim):
        super(Decoder, self).__init__()
        self.hidden_dim = hidden_dim
        self.gru = nn.GRU(hidden_dim * 2, hidden_dim, batch_first=True)  # Match encoder output dim

    def forward(self, x, hidden_state):
        d, hidden_state = self.gru(x, hidden_state)
        return d, hidden_state  # d ∈ R^(M × H)

class Pointer(nn.Module):
    def __init__(self, encoder_hidden_dim, decoder_hidden_dim):
        super(Pointer, self).__init__()
        self.W1 = nn.Linear(encoder_hidden_dim, decoder_hidden_dim)  # 2H → H
        self.W2 = nn.Linear(decoder_hidden_dim, decoder_hidden_dim)  # H → H
        self.v = nn.Linear(decoder_hidden_dim, 1, bias=False)

    def forward(self, encoder_outputs, decoder_state):
        scores = self.v(torch.tanh(self.W1(encoder_outputs) + self.W2(decoder_state)))
        attention_weights = F.softmax(scores, dim=1)  # softmax over input sequence positions
        return attention_weights

class SEGBOT(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(SEGBOT, self).__init__()
        self.encoder = Encoder(input_dim, hidden_dim)
        self.decoder = Decoder(hidden_dim)
        self.pointer = Pointer(hidden_dim * 2, hidden_dim)

    def forward(self, x, start_units):
        encoder_outputs = self.encoder(x)  # Shape: (batch, seq_len, 2H)
        decoder_hidden = torch.zeros(1, x.size(0), self.decoder.hidden_dim).to(x.device)
        decoder_inputs = encoder_outputs[:, start_units, :].unsqueeze(1)  # Shape: (batch, 1, 2H)
        decoder_outputs, _ = self.decoder(decoder_inputs, decoder_hidden)  # Shape: (batch, 1, H)
        attention_weights = self.pointer(encoder_outputs, decoder_outputs.squeeze(1))  # Shape: (batch, seq_len, 1)
        return attention_weights

    def segment_text(self, sentences, tokens, attention_weights):
        attention_weights = attention_weights.squeeze().detach().cpu().numpy()

        # Normalize attention weights
        attention_weights = (attention_weights - np.min(attention_weights)) / (np.max(attention_weights) - np.min(attention_weights)

        # Find peaks in attention scores
        peak_indices, _ = find_peaks(attention_weights, height=0.5, distance=5)  # Adjust height and distance for better segmentatio

        if len(peak_indices) == 0:
            return [" ".join(sentences)]  # Return full text if no peaks found

        segments = []
        start = 0
        for i in peak_indices:
            if i - start >= 5:  # Ensure at least 5 sentences per segment
```

```
                segment = " ".join(sentences[start:i]).strip()
                if segment:
                    segments.append(segment)
                start = i

            last_segment = " ".join(sentences[start:]).strip()
            if last_segment:
                segments.append(last_segment)  # Add last segment

            return segments if segments else None  # Return None if all segments are empty

# Model Hyperparameters
input_dim = 128  # Example input size
hidden_dim = 256  # Hidden layer size
model = SEGBOT(input_dim, hidden_dim)

# Load text file and process
file_path = "/content/transcript (15).txt"
sentences, tokens, full_text = load_text_file(file_path)

# Example Input (Dummy Tensor)
x = torch.randn(1, len(tokens), input_dim)  # Batch size of 1, sequence length based on text
start_units = 0  # Corrected variable type (integer index)
output = model(x, start_units)

# Segment the text
segments = model.segment_text(sentences, tokens, output)
if segments:
    with open("segmented_transcript_new.txt", "w", encoding="utf-8") as f:
        for i, segment in enumerate(segments):
            f.write(f"Segment {i+1}:\n{segment}\n\n")
    print("Segmented transcript saved successfully.")
else:
    print("No valid segments found. Terminating execution.")
```

```
Segmented transcript saved successfully.
```

```
Start coding or generate with AI.
```