Assessment Models Implemention & Expected Data

1. Tone and Vocal Quality (SER)

SER Output

The SER model usually classifies emotions into predefined categories such as "Happy," "Sad," "Angry," "Neutral," etc., based on the features extracted from the audio signal. These features can include pitch, energy, and formants, among others. The output might look something like this:

```
{
  "Happy": 0.7,
  "Sad": 0.1,
  "Angry": 0.1,
  "Neutral": 0.1
}
```

Here, the numbers represent the probability or confidence level that the model has in its classification. In this example, the model is 70% confident that the tone is "Happy."

Translating to Ratings To translate this into a rating system for "Tone and Vocal Quality," you could map the emotional categories to the attributes you're interested in, such as warmth, confidence, and sincerity.

For example:

- "Happy" and "Neutral" tones could be indicative of warmth.
- "Angry" and "Happy" tones could be indicative of confidence.
- "Neutral" and "Sad" tones could be indicative of sincerity.
- You could then use a weighted average formula to calculate the ratings for each attribute. Here's a simplified example:

```
# PY-2
# Filename: ser_to_rating.py
# PY-2-FN-1: Function to translate SER output to ratings
def ser_to_rating(ser_output):
    warmth = (ser_output['Happy'] * 0.6) + (ser_output['Neutral'] * 0.4)
    confidence = (ser_output['Angry'] * 0.5) + (ser_output['Happy'] * 0.5)
    sincerity = (ser_output['Neutral'] * 0.5) + (ser_output['Sad'] * 0.5)

    # Convert to 1-10 scale
    warmth_rating = round(warmth * 10)
    confidence_rating = round(confidence * 10)
    sincerity_rating = round(sincerity * 10)

    return {
        'Warmth': warmth_rating,
        'Confidence': confidence_rating,
```

```
        'Sincerity': sincerity_rating
    }
```

2. Pacing (Comprehension and Comfort)

Praat Output

Praat is a tool often used for phonetic analysis, including the measurement of speech rate. It can provide detailed information about the duration of phonemes, syllables, and pauses in the speech. A typical output might look like this:

```
{
  "PhonemeDuration": [0.1, 0.2, 0.15, ...],
  "SyllableDuration": [0.3, 0.25, 0.35, ...],
  "PauseDuration": [0.4, 0.5, 0.2, ...]
}
```

Translating to Ratings

To translate this into a rating for "Pacing," you could calculate the average duration of phonemes, syllables, and pauses, and then normalize these values to fit into a 1-10 scale. The idea is to find a pacing score that matches the complexity of the information being shared.

Here's a Python function that could perform this translation:

```
# PY-1
# Filename: praat_to_rating.py
# PY-1-FN-1: Function to translate Praat output to pacing rating
def praat_to_rating(praat_output):
    avg_phoneme_duration = sum(praat_output['PhonemeDuration']) /
len(praat_output['PhonemeDuration'])
    avg_syllable_duration = sum(praat_output['SyllableDuration']) /
len(praat_output['SyllableDuration'])
    avg_pause_duration = sum(praat_output['PauseDuration']) /
len(praat_output['PauseDuration'])

    # Normalize and weight the averages
    pacing_score = (avg_phoneme_duration * 0.3) + (avg_syllable_duration * 0.4) +
(avg_pause_duration * 0.3)

    # Convert to 1-10 scale (assuming 1 is slowest and 10 is fastest)
    pacing_rating = round(10 - (pacing_score * 10))

    return {
        'Pacing': pacing_rating
    }
```

3. Use of Silence (Thoughtfulness and Respect)

VAD Output

Voice Activity Detection (VAD) typically outputs binary flags indicating whether each segment of audio contains voice activity or silence. For example:

```
{
  "VoiceActivity": [1, 0, 1, 1, 0, 0, 1, ...]
}
```

Here, 1 indicates voice activity and 0 indicates silence.

Translating to Ratings

You could calculate the ratio of silence to voice activity and then assess how effectively silence is used for emphasis or reflection.

```
# PY-3
# Filename: vad_to_rating.py
# PY-3-FN-1: Function to translate VAD output to silence rating
def vad_to_rating(vad_output):
    total_segments = len(vad_output['VoiceActivity'])
    silence_segments = vad_output['VoiceActivity'].count(0)

    # Calculate effective use of silence
    effective_silence = (silence_segments / total_segments) * 10

    return {
        'EffectiveUseOfSilence': round(effective_silence)
    }
```

4. Audibility and Clarity (PESQ)

PESQ Output

PESQ usually outputs a Mean Opinion Score (MOS) between -0.5 and 4.5, where higher scores indicate better quality.

Translating to Ratings

The MOS can be linearly scaled to a 1-10 rating for audibility and clarity.

```
# PY-4
# Filename: pesq_to_rating.py
# PY-4-FN-1: Function to translate PESQ output to audibility and clarity rating
def pesq_to_rating(mos_score):
```

```
    # Scale MOS to 1-10
    scaled_score = (mos_score + 0.5) * 2

    return {
        'AudibilityAndClarity': round(scaled_score)
    }
```

5. Interruption Handling (Rasa)

Rasa Output

Rasa could be customized to output logs that indicate how interruptions were managed. For example, it could output a count of interruptions and how they were resolved.

Translating to Ratings

You could rate the conversation based on the number of interruptions and how well they were managed.

```
# PY-5
# Filename: rasa_to_rating.py
# PY-5-FN-1: Function to translate Rasa output to interruption handling rating
def rasa_to_rating(interruption_count, resolved_count):
    # Calculate effective interruption handling
    effective_handling = (resolved_count / interruption_count) * 10

    return {
        'InterruptionHandling': round(effective_handling)
    }
```

6. Verbal Affirmations (Wit.ai)

Wit.ai Output

Wit.ai could output a list of recognized verbal affirmations and their appropriateness in the context.

Translating to Ratings

You could rate the conversation based on the number and appropriateness of verbal affirmations.

```
# PY-6
# Filename: witai_to_rating.py
# PY-6-FN-1: Function to translate Wit.ai output to verbal affirmations rating
def witai_to_rating(affirmation_count, appropriateness_score):
    # Calculate effective verbal affirmations
    effective_affirmations = ((affirmation_count + appropriateness_score) / 2) *
10

    return {
```

```
        'VerbalAffirmations': round(effective_affirmations)
    }
```

7. Non-Verbal Sounds (AudioSet)

AudioSet Output

AudioSet could output a list of recognized non-verbal sounds like sighs, laughter, or pauses, and their impact on the conversation's tone and meaning.

Translating to Ratings

You could rate the conversation based on the impact of these non-verbal sounds.

```
# PY-7
# Filename: audioset_to_rating.py
# PY-7-FN-1: Function to translate AudioSet output to non-verbal sounds rating
def audioset_to_rating(impact_score):
    # Scale impact score to 1-10
    scaled_score = impact_score * 10

    return {
        'NonVerbalSounds': round(scaled_score)
    }
```

8. Call Technical Quality (WebRTC)

WebRTC Output

WebRTC could output metrics like latency, packet loss, and jitter, which could be used to assess call quality.

Translating to Ratings

You could rate the call based on these technical metrics.

```
# PY-8
# Filename: webrtc_to_rating.py
# PY-8-FN-1: Function to translate WebRTC output to call technical quality rating
def webrtc_to_rating(latency, packet_loss, jitter):
    # Calculate effective call quality
    effective_quality = 10 - ((latency + packet_loss + jitter) / 3)

    return {
        'CallTechnicalQuality': round(effective_quality)
    }
```