Complete Guide to Goodness of Pronunciation Calculation

Goodness of Pronunciation (GOP) serves as the mathematical backbone of modern pronunciation assessment systems, transforming acoustic speech signals into quantitative measures that rival human expert evaluations. (Medium) (ResearchGate) This comprehensive guide explores GOP's theoretical foundations, practical implementation, and cutting-edge advances that are reshaping automatic pronunciation training systems worldwide.

What GOP measures and why it matters

GOP functions as a digital pronunciation judge, calculating numerical scores that quantify how closely a spoken phoneme matches expected native pronunciation patterns. Introduced by Witt and Young in 2000, GOP addresses a fundamental challenge in language learning: providing objective, consistent pronunciation feedback that scales beyond human instructors. (PubMed Central +7)

The core innovation lies in GOP's **likelihood-based scoring mechanism**. Think of it as comparing fingerprints - GOP measures how well a speaker's acoustic "fingerprint" for each phoneme matches the expected native speaker template. (arXiv) This mathematical approach enables automated systems to detect mispronunciations with **80-95% accuracy** while achieving **0.4-0.7 correlation** with human expert ratings.

GOP operates at the **phoneme level**, providing granular feedback that pinpoints specific pronunciation errors. (arXiv +5) This precision makes it invaluable for Computer-Assisted Language Learning (CALL) systems, (ScienceDirect) speech therapy applications, and pronunciation training platforms (arXiv) serving millions of language learners globally.

Mathematical foundations and core formulations

Original likelihood-based GOP

The foundational GOP equation computes pronunciation quality through likelihood ratios: (PubMed Central +3)

```
GOP(q) = log P(O(q)|q) - max_k log P(O(q)|k)
```

Where:

- `O(q)` represents acoustic observations for phoneme q PubMed Central
- `P(O(q)|q)` is the likelihood of features given target phoneme PubMed Central
- `max_k P(O(q)|k)` represents maximum likelihood across competing phonemes PubMed Central

Think of this as a contest: the target phoneme must "win" against all competitors. Higher GOP scores indicate the target phoneme is a clear winner, while lower scores suggest confusion with other sounds.

Modern deep neural network GOP

Contemporary systems utilize posterior probabilities from neural networks: arXiv ResearchGate

```
GOP_DNN(q) = -log(1/T ∑_{t=1}^T P(q|x_t))
```

Where:

- `P(q|x_t)` is the softmax probability of phoneme q at frame t arXiv
- `T` represents total frames in the phoneme segment arXiv
- The negative logarithm converts probabilities to additive scores GitHub

Advanced logit-based formulations (2024-2025)

Recent research addresses softmax limitations through raw logit processing: arXiv

```python
```

```
# Maximum logit GOP
GOP_MaxLogit = max(logits_for_target_phoneme)

# Mean margin GOP
GOP_Margin = mean(target_logits - max(competing_logits))

# Logit variance GOP
GOP_VarLogit = variance(target_logits_across_frames)
```

These formulations avoid the **overconfidence problem** in softmax probabilities, providing more discriminative pronunciation assessment. arXiv

Step-by-step algorithmic process

Phase 1: Audio preprocessing and feature extraction

```python
import librosa
import numpy as np
from scipy.signal import get_window

def extract_mfcc_features(audio_file, sr=16000, n_mfcc=13):
    """Extract MFCC features for GOP calculation"""
    # Load audio at 16kHz sampling rate
    y, sr = librosa.load(audio_file, sr=sr)

    # Extract MFCC features with delta and delta-delta
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
    delta_mfccs = librosa.feature.delta(mfccs)
    delta2_mfccs = librosa.feature.delta(mfccs, order=2)

    # Combine features (39 dimensions total)
    features = np.vstack([mfccs, delta_mfccs, delta2_mfccs])
    return features.T  # Shape: (n_frames, n_features)
```

Phase 2: Forced alignment for phoneme boundaries

```python
def perform_forced_alignment(audio_features, transcript, acoustic_model):
    """
    Align phonemes to acoustic frames using Viterbi algorithm
    This is conceptually like finding the best path through a maze
    where each room represents a phoneme state
    """
    # Simplified alignment process (actual implementation uses Kaldi/ESPnet)
    phoneme_boundaries = []

    for phoneme in transcript:
        # Viterbi decoding finds optimal state sequence
        start_frame, end_frame = viterbi_align(
            features=audio_features,
            phoneme=phoneme,
            acoustic_model=acoustic_model
        )

        phoneme_boundaries.append({
            'phoneme': phoneme,
            'start': start_frame,
            'end': end_frame
        })

    return phoneme_boundaries
```

Phase 3: GOP calculation with neural network posteriors

```python
```

```python
def calculate_gop_scores(features, alignments, acoustic_model):
    """
    Calculate GOP scores using DNN posterior probabilities
    """
    gop_scores = []

    for alignment in alignments:
        phoneme = alignment['phoneme']
        start = alignment['start']
        end = alignment['end']

        # Extract features for this phoneme segment
        segment_features = features[start:end+1]

        # Get posterior probabilities from DNN
        posteriors = acoustic_model.predict(segment_features)

        # Calculate mean posterior for target phoneme
        phoneme_id = phoneme_to_id[phoneme]
        target_posteriors = posteriors[:, phoneme_id]

        # GOP score is negative log of mean posterior
        mean_posterior = np.mean(target_posteriors)
        gop_score = -np.log(mean_posterior + 1e-10)  # Add small epsilon

        gop_scores.append({
            'phoneme': phoneme,
            'gop_score': gop_score,
            'duration': end - start + 1
        })

    return gop_scores
```

Complete GOP pipeline implementation

```python
```

```python
class GOPCalculator:
    def __init__(self, acoustic_model_path, pronunciation_dict):
        self.acoustic_model = self.load_model(acoustic_model_path)
        self.pronunciation_dict = pronunciation_dict

    def calculate_utterance_gop(self, audio_file, transcript):
        """
        Complete GOP calculation pipeline
        """
        # Step 1: Feature extraction
        features = self.extract_mfcc_features(audio_file)

        # Step 2: Convert transcript to phonemes
        phonemes = self.text_to_phonemes(transcript)

        # Step 3: Forced alignment
        alignments = self.perform_forced_alignment(features, phonemes)

        # Step 4: GOP calculation
        gop_scores = self.calculate_gop_scores(features, alignments)

        # Step 5: Normalize and return results
        return self.normalize_gop_scores(gop_scores)

    def text_to_phonemes(self, text):
        """Convert text to phoneme sequence using pronunciation dictionary"""
        words = text.lower().split()
        phonemes = []
        for word in words:
            if word in self.pronunciation_dict:
                phonemes.extend(self.pronunciation_dict[word])
        return phonemes

    def normalize_gop_scores(self, raw_scores):
        """Apply duration normalization and scaling"""
        normalized_scores = []
        for score_info in raw_scores:
            # Duration normalization (already applied in calculation)
            # Additional scaling for human interpretability
```

```
            normalized_score = min(max(score_info['gop_score'], 0), 10)
            normalized_scores.append({
                'phoneme': score_info['phoneme'],
                'gop_score': normalized_score,
                'quality': self.score_to_quality(normalized_score)
            })
        return normalized_scores

    def score_to_quality(self, score):
        """Convert numerical GOP to quality labels"""
        if score < 2.0:
            return "Excellent"
        elif score < 4.0:
            return "Good"
        elif score < 6.0:
            return "Fair"
        else:
            return "Needs Improvement"
```

Technical concepts through metaphors and analogies

GOP as a speech recognition confidence meter

Imagine GOP as a specialized confidence meter for pronunciation. Like a metal detector that beeps louder when finding the target metal, GOP produces higher "confidence beeps" (lower numerical scores) when the acoustic signal strongly matches the target phoneme. When the signal is ambiguous or matches competing phonemes, the confidence decreases.

Likelihood ratios as pronunciation contests

The likelihood ratio concept resembles **a pronunciation beauty contest** where each phoneme competes for the best explanation of observed acoustic features. The target phoneme must not only score well but must clearly outperform all competitors. GitHub A strong GOP score means the target phoneme is an undisputed winner, while weak scores indicate a close contest with potential mispronunciation.

Posterior probabilities as voting systems

DNN-based GOP operates like a democratic voting system where each acoustic frame "votes" for the most likely phoneme. The posterior probability represents the strength of votes for each candidate. arXiv High posterior probabilities for the target phoneme indicate unanimous support, while low probabilities suggest split votes among competing phonemes.

Forced alignment as timeline synchronization

Forced alignment functions like synchronizing a movie soundtrack with video. The acoustic signal is the "video" and the phoneme transcript is the "soundtrack." The alignment process finds the optimal synchronization where each phoneme aligns with its corresponding acoustic segment, enabling precise GOP calculation for each sound. arXiv

Advanced GOP variants and approaches

Context-aware GOP (CaGOP)

CaGOP addresses traditional GOP limitations by incorporating phonemic context: ResearchGate +2

```python
```

```
def calculate_context_aware_gop(features, alignments, acoustic_model):
    """
    Calculate GOP with transition and duration factors
    """
    base_gop_scores = calculate_basic_gop(features, alignments, acoustic_model)

    contextual_scores = []
    for i, score_info in enumerate(base_gop_scores):
        # Calculate transition factor based on neighboring phonemes
        prev_phoneme = alignments[i-1]['phoneme'] if i > 0 else None
        next_phoneme = alignments[i+1]['phoneme'] if i < len(alignments)-1 else None

        transition_factor = calculate_transition_factor(
            score_info['phoneme'], prev_phoneme, next_phoneme
        )

        # Calculate duration factor using self-attention
        duration_factor = calculate_duration_factor(
            score_info['duration'], score_info['phoneme']
        )

        # Combine factors for context-aware score
        context_gop = score_info['gop_score'] * transition_factor * duration_factor

        contextual_scores.append({
            'phoneme': score_info['phoneme'],
            'gop_score': context_gop,
            'transition_factor': transition_factor,
            'duration_factor': duration_factor
        })

    return contextual_scores
```

Logit-based GOP for improved discrimination

```
python
```

```python
def calculate_logit_based_gop(features, alignments, dnn_model):
    """
    Calculate GOP using raw logits instead of softmax probabilities
    """
    gop_variants = []

    for alignment in alignments:
        segment_features = extract_segment(features, alignment)

        # Get raw logits from DNN (before softmax)
        logits = dnn_model.get_logits(segment_features)  # Shape: (n_frames, n_phonemes)

        target_phoneme_id = phoneme_to_id[alignment['phoneme']]
        target_logits = logits[:, target_phoneme_id]

        # Maximum logit GOP
        max_logit_gop = np.max(target_logits)

        # Mean margin GOP (target vs best competitor)
        competing_logits = np.delete(logits, target_phoneme_id, axis=1)
        max_competing = np.max(competing_logits, axis=1)
        margin_gop = np.mean(target_logits - max_competing)

        # Logit variance GOP
        variance_gop = np.var(target_logits)

        gop_variants.append({
            'phoneme': alignment['phoneme'],
            'max_logit_gop': max_logit_gop,
            'margin_gop': margin_gop,
            'variance_gop': variance_gop,
            'combined_gop': 0.6 * margin_gop + 0.4 * max_logit_gop
        })

    return gop_variants
```

Alignment-free GOP using CTC

python

```python
def calculate_alignment_free_gop(audio_features, target_phonemes, ctc_model):
    """
    Calculate GOP without forced alignment using CTC-based model
    """
    # Get CTC probabilities for all possible alignments
    ctc_probs = ctc_model.predict(audio_features)  # Shape: (n_frames, n_phonemes + 1)

    # Calculate forward-backward probabilities for target sequence
    target_sequence_prob = ctc_forward_backward(ctc_probs, target_phonemes)

    # Calculate competing sequence probabilities
    competing_probs = []
    for alternative_seq in generate_phoneme_alternatives(target_phonemes):
        alt_prob = ctc_forward_backward(ctc_probs, alternative_seq)
        competing_probs.append(alt_prob)

    # GOP as log ratio of target vs best alternative
    best_competing_prob = max(competing_probs)
    alignment_free_gop = np.log(target_sequence_prob / best_competing_prob)

    return alignment_free_gop
```

arXiv

Implementation considerations and practical challenges

Computational efficiency optimization

Real-time processing demands careful optimization. Modern DNN-based GOP systems require approximately **1-2x real-time processing**, meaning a 10-second audio clip takes 10-20 seconds to process. ScienceDirect Key optimization strategies include:

1. **Model pruning**: Reducing neural network size by removing less important connections
2. **Quantization**: Converting 32-bit floating point to 8-bit integer calculations
3. **Batch processing**: Processing multiple utterances simultaneously
4. **GPU acceleration**: Leveraging parallel processing for matrix operations

```python
def optimize_gop_for_realtime(model_path, target_latency_ms=500):
    """
    Optimize GOP model for real-time processing
    """
    # Load and quantize model
    original_model = load_acoustic_model(model_path)
    quantized_model = quantize_model(original_model, precision='int8')

    # Implement sliding window processing for streaming
    window_size = int(target_latency_ms * 16)  # 16kHz sampling rate

    class StreamingGOPCalculator:
        def __init__(self, model):
            self.model = model
            self.feature_buffer = []

        def process_audio_chunk(self, audio_chunk):
            # Extract features for current chunk
            chunk_features = extract_mfcc_features(audio_chunk)
            self.feature_buffer.extend(chunk_features)

            # Process if buffer has sufficient frames
            if len(self.feature_buffer) >= window_size:
                # Calculate GOP for current window
                window_features = np.array(self.feature_buffer[-window_size:])
                gop_score = self.calculate_chunk_gop(window_features)
                return gop_score

            return None

    return StreamingGOPCalculator(quantized_model)
```

Data imbalance and threshold optimization

Most phonemes are pronounced correctly even by poor speakers, creating severe class imbalance. GitHub This challenge requires sophisticated threshold optimization:

```python
def optimize_gop_thresholds(gop_scores, human_labels, optimization_metric='mcc'):
    """
    Optimize GOP decision thresholds using cross-validation
    """
    from sklearn.metrics import matthews_corrcoef, roc_auc_score
    from sklearn.model_selection import cross_val_score

    best_thresholds = {}

    for phoneme in set(score_info['phoneme'] for score_info in gop_scores):
        phoneme_scores = [s['gop_score'] for s in gop_scores
                        if s['phoneme'] == phoneme]
        phoneme_labels = [l for s, l in zip(gop_scores, human_labels)
                        if s['phoneme'] == phoneme]

        # Grid search for optimal threshold
        thresholds = np.linspace(min(phoneme_scores), max(phoneme_scores), 100)
        best_score = -1
        best_threshold = thresholds[0]

        for threshold in thresholds:
            predictions = [1 if score > threshold else 0 for score in phoneme_scores]

            if optimization_metric == 'mcc':
                score = matthews_corrcoef(phoneme_labels, predictions)
            elif optimization_metric == 'auc':
                score = roc_auc_score(phoneme_labels, predictions)

            if score > best_score:
                best_score = score
                best_threshold = threshold

        best_thresholds[phoneme] = best_threshold

    return best_thresholds
```

Cross-linguistic adaptation challenges

Language-specific phoneme systems require careful model adaptation. Different languages have varying phoneme inventories, coarticulation patterns, and timing characteristics:

python

```python
def adapt_gop_for_language_pair(base_model, l1_language, l2_language, adaptation_data):
    """
    Adapt GOP model for specific L1-L2 language pair
    """
    # Define common mispronunciation patterns for this language pair
    substitution_patterns = {
        ('mandarin', 'english'): {
            '/r/': ['/l/', '/w/'],  # R-L confusion
            '/θ/': ['/s/', '/f/'],  # TH-S/F substitution
            '/ð/': ['/z/', '/d/']   # TH-Z/D substitution
        }
    }

    # Fine-tune acoustic model on L1-specific data
    adapted_model = fine_tune_acoustic_model(
        base_model=base_model,
        adaptation_data=adaptation_data,
        target_phonemes=get_l2_phonemes(l2_language),
        confusion_patterns=substitution_patterns.get((l1_language, l2_language), {})
    )

    # Adjust GOP calculation to weight known confusions
    def calculate_adapted_gop(features, alignments):
        base_scores = calculate_basic_gop(features, alignments, adapted_model)

        adjusted_scores = []
        for score_info in base_scores:
            phoneme = score_info['phoneme']
            base_score = score_info['gop_score']

            # Apply confusion-specific adjustments
            if phoneme in substitution_patterns.get((l1_language, l2_language), {}):
                confusion_penalty = calculate_confusion_penalty(phoneme, features)
                adjusted_score = base_score + confusion_penalty
            else:
                adjusted_score = base_score

            adjusted_scores.append({
                'phoneme': phoneme,
```

```python
            'gop_score': adjusted_score,
            'adaptation_applied': phoneme in substitution_patterns.get((l1_language, l2_language), {})
        })

    return adjusted_scores

return calculate_adapted_gop
```

GOP relationship to other pronunciation metrics

Comparison with confidence scores

Traditional ASR confidence scores measure recognition certainty, while GOP specifically targets pronunciation quality. Key differences include:

- **Scope**: Confidence scores assess overall recognition reliability; GOP focuses on phoneme-level pronunciation accuracy
- **Training data**: Confidence models use general speech; GOP requires pronunciation-specific training
- **Granularity**: Confidence operates at word/utterance level; GOP provides phoneme-level detail

```python
```

```python
def compare_gop_with_confidence(audio_file, transcript, asr_model, gop_calculator):
    """
    Compare GOP scores with traditional ASR confidence measures
    """
    # Get ASR confidence scores
    asr_result = asr_model.recognize(audio_file)
    confidence_scores = asr_result['confidence']

    # Calculate GOP scores
    gop_scores = gop_calculator.calculate_utterance_gop(audio_file, transcript)

    # Analyze correlation
    word_level_gop = aggregate_phoneme_to_word_gop(gop_scores, transcript)
    correlation = np.corrcoef(confidence_scores, word_level_gop)[0, 1]

    return {
        'correlation': correlation,
        'confidence_scores': confidence_scores,
        'gop_scores': word_level_gop,
        'analysis': interpret_correlation(correlation)
    }
```

Integration with multi-dimensional assessment

Modern pronunciation systems combine GOP with complementary metrics: IEEE Xplore

python

```python
def calculate_comprehensive_pronunciation_score(audio_file, transcript):
    """
    Calculate multi-dimensional pronunciation assessment
    """
    # GOP for accuracy
    gop_scores = calculate_gop_scores(audio_file, transcript)
    accuracy_score = np.mean([s['gop_score'] for s in gop_scores])

    # Fluency assessment (speaking rate, pauses)
    fluency_score = calculate_fluency_score(audio_file)

    # Prosody assessment (rhythm, stress, intonation)
    prosody_score = calculate_prosody_score(audio_file, transcript)

    # Completeness (missing sounds, extra sounds)
    completeness_score = calculate_completeness_score(audio_file, transcript)

    # Weighted combination
    comprehensive_score = (
        0.4 * accuracy_score +
        0.25 * fluency_score +
        0.2 * prosody_score +
        0.15 * completeness_score
    )

    return {
        'overall_score': comprehensive_score,
        'accuracy': accuracy_score,
        'fluency': fluency_score,
        'prosody': prosody_score,
        'completeness': completeness_score
    }
```

Real-world applications and deployment

Computer-Assisted Language Learning (CALL)

Commercial applications like SpeechAce serve major publishers and educational platforms with millions of users. speechace Medium GOP enables:

- **Interactive pronunciation feedback**: Real-time identification of pronunciation errors
- **Progress tracking**: Longitudinal monitoring of pronunciation improvement
- **Adaptive difficulty**: Adjusting exercises based on individual GOP performance patterns

Speech therapy and clinical applications

Medical applications extend GOP beyond language learning: PubMed Central +2

python

```python
def clinical_pronunciation_assessment(patient_audio, reference_patterns):
    """
    Adapt GOP for clinical speech assessment
    """
    # Calculate standard GOP scores
    gop_scores = calculate_gop_scores(patient_audio, reference_patterns)

    # Apply clinical interpretation thresholds
    clinical_assessment = []
    for score in gop_scores:
        severity = categorize_clinical_severity(score['gop_score'])
        clinical_assessment.append({
            'phoneme': score['phoneme'],
            'severity': severity,
            'therapy_priority': assign_therapy_priority(severity),
            'recommended_exercises': suggest_exercises(score['phoneme'], severity)
        })

    return clinical_assessment
```

Commercial API integration

Enterprise deployment requires robust, scalable architectures:

python

```python
class GOPService:
    """
    Production-ready GOP calculation service
    """
    def __init__(self, model_config):
        self.models = load_multi_language_models(model_config)
        self.cache = initialize_result_cache()

    async def assess_pronunciation(self, audio_data, transcript, language_pair):
        """
        Async GOP calculation with caching and error handling
        """
        try:
            # Check cache first
            cache_key = generate_cache_key(audio_data, transcript, language_pair)
            if cache_key in self.cache:
                return self.cache[cache_key]

            # Validate input
            if not self.validate_input(audio_data, transcript):
                raise ValueError("Invalid input data")

            # Calculate GOP
            model = self.models[language_pair]
            gop_result = await self.calculate_gop_async(audio_data, transcript, model)

            # Cache and return result
            self.cache[cache_key] = gop_result
            return gop_result

        except Exception as e:
            logger.error(f"GOP calculation failed: {str(e)}")
            return self.generate_fallback_response(e)
```

Recent advances and future directions

Self-supervised learning revolution (2022-2025)

Foundation models like Wav2vec2.0 and HuBERT are transforming GOP calculation by leveraging massive unlabeled speech data. These models learn rich acoustic representations without requiring pronunciation annotations. ResearchGate +2

```python
def calculate_ssl_based_gop(audio_file, transcript, ssl_model='wav2vec2'):
    """
    Calculate GOP using self-supervised learning representations
    """
    # Extract SSL features
    if ssl_model == 'wav2vec2':
        ssl_features = wav2vec2_feature_extractor(audio_file)
    elif ssl_model == 'hubert':
        ssl_features = hubert_feature_extractor(audio_file)

    # Fine-tune on pronunciation task
    pronunciation_model = fine_tune_ssl_for_gop(ssl_features, transcript)

    # Calculate enhanced GOP scores
    enhanced_gop = pronunciation_model.calculate_gop(ssl_features)

    return enhanced_gop
```

Transformer-based multi-aspect assessment

GOPT (Goodness Of Pronunciation Transformer) represents the current state-of-the-art, achieving **0.742 sentence-level correlation** with human ratings. This system simultaneously assesses accuracy, fluency, prosody, and completeness. GitHub +2

Large language model integration (2024-2025)

LLM-based pronunciation systems provide detailed explanations alongside numerical scores: arXiv

```python
```

```python
def llm_enhanced_pronunciation_feedback(gop_scores, audio_features, text):
    """
    Generate detailed pronunciation feedback using large language models
    """
    # Prepare context for LLM
    pronunciation_context = {
        'gop_scores': gop_scores,
        'acoustic_analysis': analyze_acoustic_patterns(audio_features),
        'target_text': text,
        'error_patterns': identify_error_patterns(gop_scores)
    }

    # Generate detailed feedback
    llm_prompt = f"""
    Analyze the pronunciation assessment data and provide detailed feedback:
    GOP Scores: {pronunciation_context['gop_scores']}
    Error Patterns: {pronunciation_context['error_patterns']}

    Provide:
    1. Specific pronunciation errors identified
    2. Suggested improvement strategies
    3. Practice exercises tailored to these errors
    """

    detailed_feedback = llm_model.generate(llm_prompt)

    return {
        'numerical_scores': gop_scores,
        'detailed_feedback': detailed_feedback,
        'practice_recommendations': extract_recommendations(detailed_feedback)
    }
```

Performance benchmarks and future outlook

Current GOP systems achieve impressive performance metrics:

- **Phone-level accuracy**: 80-85% for binary correct/incorrect classification
- **Human correlation**: 0.65-0.75 Pearson correlation with expert ratings arXiv GitHub

- **Real-time processing**: 1-2x real-time latency for DNN-based systems

Emerging trends point toward even more sophisticated systems:

- **Multimodal assessment**: Incorporating visual lip movement analysis
- **Personalized adaptation**: Systems that adapt to individual learner characteristics
- **Uncertainty quantification**: Providing confidence intervals for GOP scores arXiv
- **Edge computing optimization**: Enabling high-quality GOP on mobile devices

The evolution from simple likelihood ratios to sophisticated neural architectures demonstrates GOP's continuing relevance in automatic pronunciation assessment. GitHub ResearchGate As these systems become more accurate, efficient, and interpretable, they promise to democratize high-quality pronunciation training globally, making expert-level feedback accessible to learners regardless of geographic location or economic circumstances.

GOP's fundamental insight remains powerful: by quantifying the acoustic evidence for pronunciation quality, these systems bridge the gap between subjective human judgment and objective computational assessment, PubMed Central enabling scalable, consistent pronunciation training that adapts to each learner's unique needs.