

[ChatGPT-Log]

Datum: 2025-10-17

Uhrzeit: 14:03

Thema/Ziel: Wiener Filter

Prompt: Erkläre mir Wiener Filterung im Zusammenhang der Akustik

Reply:

Was ist die Wiener-Filterung?

Der **Wiener-Filter** ist ein **optimales lineares Filterverfahren**, das entwickelt wurde, um ein **Signal aus einem verrauschten Signal** zu schätzen.

Er wurde von **Norbert Wiener** (1940er Jahre) entwickelt und basiert auf der **Minimierung des mittleren quadratischen Fehlers (MSE)** zwischen dem gefilterten Signal und dem ursprünglichen (unverrauschten) Signal.

Grundidee

Angenommen, du hast ein gemessenes Signal:

$$x(t) = s(t) + n(t)$$

mit

- $s(t)$: das **gesuchte (saubere)** Sprachsignal
- $n(t)$: **Rauschen**

Der Wiener-Filter möchte nun eine **Schätzung** $\hat{s}(t)$ finden, die $s(t)$ möglichst gut entspricht.

Mathematische Idee (Frequenzbereich)

Im Frequenzbereich lautet der Wiener-Filter:

$$H(f) = \frac{S_{ss}(f)}{S_{ss}(f) + S_{nn}(f)}$$

mit

- $S_{ss}(f)$: Leistungsspektrum des Sprachsignals
- $S_{nn}(f)$: Leistungsspektrum des Rauschens

Das gefilterte Signal ergibt sich zu:

$$\hat{s}(f) = H(f) \cdot X(f)$$

Intuition

- Wenn an einer Frequenz **viel Sprachenergie** und **wenig Rauschen** vorhanden ist $\rightarrow H(f) \approx 1$: diese Frequenzen bleiben erhalten.

- Wenn an einer Frequenz **viel Rauschen** und **wenig Sprache** ist $\rightarrow H(f) \approx 0$: diese Frequenzen werden gedämpft.

→ Der Wiener-Filter **passt sich adaptiv** dem Signal-zu-Rausch-Verhältnis (SNR) an.

Anwendung in der Akustik

In der **Sprachverbesserung** oder **Hörtechnik** wird der Wiener-Filter oft verwendet, um:

- **Rauschen zu unterdrücken** (z. B. Lüfter-, Straßen- oder Mikrofonrauschen)
- **Verständlichkeit** von Sprache zu erhöhen
- **Vorverarbeitung** für ASR-Systeme (Automatic Speech Recognition)

Beispiel:

1. Rauschanteil in einer stillen Phase schätzen
2. Sprach- und Rauschleistungsspektren bestimmen
3. Wiener-Filter anwenden
4. Rücktransformation in den Zeitbereich

Prompt: Was ist SNR?

Reply:

Super Frage — das ist zentral, um die **Wiener-Filterung** richtig zu verstehen.

Lass uns das Schritt für Schritt aufbauen 

◆ 1. Ziel: Rauschunterdrückung durch Gewichtung im Frequenzbereich

Wir haben ein verrauschtes Signal

$$x(t) = s(t) + n(t)$$

und wollen aus $x(t)$ eine möglichst gute Schätzung $\hat{s}(t)$ für das saubere Signal $s(t)$ berechnen.

Im Frequenzbereich:

$$X(f) = S(f) + N(f)$$

Der Wiener-Filter $H(f)$ bestimmt, **wie stark jede Frequenz** des Signals durchgelassen wird:

$$\hat{S}(f) = H(f) \cdot X(f)$$

◆ 2. Wiener-Filter-Formel

Die klassische Form des Wiener-Filters lautet:

$$H(f) = \frac{S_{ss}(f)}{S_{ss}(f) + S_{nn}(f)}$$

oder, wenn man das **Signal-zu-Rausch-Verhältnis (SNR)** benutzt:

$$H(f) = \frac{1}{1 + \frac{1}{\text{SNR}(f)}}$$

◆ **3. Was ist das SNR?**

Das **Signal-to-Noise Ratio (SNR)** beschreibt das Verhältnis zwischen **Signalenergie** und **Rauschenergie**.

Im Frequenzbereich:

$$\text{SNR}(f) = \frac{S_{ss}(f)}{S_{nn}(f)}$$

👉 Ein hoher SNR bedeutet:

- Signal dominiert $\rightarrow S_{ss} \gg S_{nn}$
- $H(f) \approx 1$: Frequenz wird **durchgelassen**

👉 Ein niedriger SNR bedeutet:

- Rauschen dominiert $\rightarrow S_{nn} \gg S_{ss}$
 - $H(f) \approx 0$: Frequenz wird **gedämpft**
-

◆ **4. Praktische Berechnung im Short-Time Fourier Transform (STFT)**

In der Praxis arbeitet man meist **zeitabhängig**, also frameweise in der STFT-Domäne:

$$X(k, m) = S(k, m) + N(k, m)$$

- k : Frequenzindex
- m : Zeitindex (Frame)

Dann gilt:

$$H(k, m) = \frac{S_{ss}(k, m)}{S_{ss}(k, m) + S_{nn}(k, m)}$$

Da S_{ss} (saubere Sprachleistung) nicht direkt bekannt ist, schätzt man sie aus dem **SNR**:

$$H(k, m) = \frac{\text{SNR}(k, m)}{\text{SNR}(k, m) + 1}$$

Prompt: Was würdest du zur Umsetzung empfehlen? Python, Matlab, C++?

Reply:

Kurzfassung: **Zum schnellen Testen → Python.**

Für „saubere“ Uni-Demos → MATLAB.

Für Echtzeit/Embedded/Produkt → C++.

Warum?

- **Python (Empfehlung fürs Testen)**

- Schnellste Iteration: NumPy/SciPy, stft/istft, Plotten mit Matplotlib.
 - Viele Audio-Tools (librosa, soundfile) + Metriken (pystoi, pesq/visqol).
 - Leicht Skripte bauen, Batch-Tests und CSV/Notebooks.
 - Nachteil: nicht echtzeitfest out-of-the-box, langsamer als C++.
- **MATLAB**
 - Top für Lehre/Demos: stft/istft, Audio Toolbox, schöne Plots in 5 Min.
 - Deterministisches Umfeld (keine „Dependency-Hölle“).
 - Nachteil: Lizenz/Ökosystem, weniger geeignet für spätere Produktion.
 - **C++**
 - Beste Performance + Echtzeit (Hörgeräte, Mobile, DSP auf ARM/NEON).
 - Zugriff auf FFTW/KissFFT, JUCE, RtAudio/PortAudio, dr_wav/sndfile.
 - Nachteil: Entwicklungszeit höher, Memory/RT-Safety beachten.

[ChatGPT-Log]

Datum: 2025-10-20

Uhrzeit: 11:20

Thema/Ziel: Spektrale subtraktion

Prompt: Wie funktioniert die spektrale subtraktion?

Wie wird die berechnet?

Reply:

Sehr gute Frage — die **Spektrale Subtraktion** ist eines der ältesten und einfachsten Verfahren zur **Sprachverbesserung (Speech Enhancement)** und Rauschunterdrückung. Ich erkläre dir erst das Prinzip und dann, wie die Berechnung konkret abläuft.

⌚ Grundidee

Wenn du ein verrausches Signal $x(t)$ hast:

$$x(t) = s(t) + n(t)$$

mit

- $s(t)$: Sprachsignal
- $n(t)$: Rauschsignal

Dann gilt im **Frequenzbereich**:

$$X(\omega) = S(\omega) + N(\omega)$$

Da die Energie (bzw. Leistung) additiv ist, gilt näherungsweise:

$$|X(\omega)|^2 = |S(\omega)|^2 + |N(\omega)|^2$$

☞ Idee: Wenn man das Rauschspektrum $|N(\omega)|$ schätzen kann (z. B. in Sprachpausen), dann kann man es vom gemessenen Spektrum $|X(\omega)|$ abziehen, um $|S(\omega)|$ zu schätzen.

Berechnungsschritte

1. Zerlegung in Frames und STFT

- Das Signal $x(t)$ wird in überlappende Frames (z. B. 20–30 ms) aufgeteilt.
- Für jedes Frame wird die **Kurzzeit-Fourier-Transformation (STFT)** berechnet:

$$X(k, m) = \text{STFT}\{x(t)\}$$

wobei k Frequenzindex und m Frameindex ist.

2. Rauschleistungsabschätzung

- In Sprachpausen (oder durch ein Sprachaktivitätserkennungsmodul, VAD) wird das mittlere Rauschspektrum geschätzt:

$$\hat{P}_N(k) = E[|N(k, m)|^2]$$

3. Subtraktion im Betragsquadrat-Spektrum

- Für jedes Frame wird das bereinigte Spektrum geschätzt:

$$\hat{P}_S(k, m) = |X(k, m)|^2 - \alpha \hat{P}_N(k)$$

wobei

- α ein **Over-Subtraction-Faktor** ist (typ. 1–2), um Restrauschen zu vermeiden.
- negative Werte werden **geklippt**, d. h.:

$$\hat{P}_S(k, m) = \max(\hat{P}_S(k, m), \beta \hat{P}_N(k))$$

mit kleinem β (Residual Noise Floor, z. B. 0.01–0.1).

4. Amplitude wiederherstellen

- Die Amplitude wird aus der Wurzel gewonnen:

$$|\hat{S}(k, m)| = \sqrt{\hat{P}_S(k, m)}$$

5. Phase beibehalten

- Da die Phase schwer zu schätzen ist, nimmt man einfach die Phase des verrauschten Signals:

$$\angle \hat{S}(k, m) = \angle X(k, m)$$

6. Inverse STFT (Rekonstruktion)

- Schließlich:

$$\hat{s}(t) = \text{iSTFT}\{|\hat{S}(k, m)| e^{j\angle X(k, m)}\}$$

Zusammenfassung als Algorithmus

Für jedes Frame m:

$$X = \text{STFT}(x)$$

$$|X|^2 = \text{Magnitudenspektrum}^2$$

Estimate noise power $|N|^2$ (aus Sprachpausen)

$$|S|^2 = |X|^2 - \alpha * |N|^2$$

$$|S|^2 = \max(|S|^2, \beta * |N|^2)$$

Recombine: $\hat{S} = |S| * \exp(j * \text{angle}(X))$

Ende

Inverse STFT über alle Frames

Parameterwahl (typische Werte)

Parameter	Bedeutung	Typischer Wert
Framegröße	FFT-Fensterlänge	256–1024
Überlappung	Anteil	50–75 %
α (Over-Subtraction)	Stärke der Rauschunterdrückung 1–2	
β (Spectral Floor)	Restgeräuschbegrenzung	0.01–0.1

[DeepSeek-Log]

Datum: 2025-10-20

Uhrzeit: 16:10

Thema/Ziel: Wiener Filter

Prompt: Erkläre mir Wiener Filterung im Bereich Audio. Vor allem die mathematische Berechnung

Reply:

Absolut! Die Wiener-Filterung ist ein fundamentales Konzept der digitalen Signalverarbeitung, besonders im Audio-Bereich. Ich erkläre es dir Schritt für Schritt, mit einem starken Fokus auf die mathematische Berechnung und deren intuitive Bedeutung.

1. Das Grundproblem: Rauschen

Stell dir vor, du hast ein Audiosignal, das durch Rauschen verunreinigt ist.

- $x[n]$: Das ursprüngliche, saubere Audiosignal (das wir nicht haben).
- $d[n]$: Das störende Rauschen (z.B. Hintergrundrauschen, Strombrummen).
- $y[n]$: Das gemessene, verrauschte Signal, das uns vorliegt.

$$y[n] = x[n] + d[n]$$

Unser Ziel ist es, einen Filter $h[n]$ zu finden, den wir auf $y[n]$ anwenden, um eine Schätzung $\hat{x}[n]$ des ursprünglichen Signals $x[n]$ zu erhalten. Je näher $\hat{x}[n]$ an $x[n]$ herankommt, desto besser.

$$\hat{x}[n] = h[n] * y[n]$$

(Dabei ist $*$ der Faltungsoperator).

2. Das Ziel des Wiener-Filters: Minimierung des Fehlers

Der "Fehler" unserer Schätzung ist die Differenz zwischen dem wahren Signal und unserer Schätzung:

$$e[n] = x[n] - \hat{x}[n]$$

Ein naiver Ansatz wäre, den mittleren Fehler $\mathbb{E}\{e[n]\}$ zu minimieren. Das funktioniert nicht, da sich positive und negative Fehler aufheben können. Die bessere Idee ist es, die **mittlere quadratische Abweichung** (Mean Square Error, MSE) zu minimieren.

Das ist das Kernkriterium des Wiener-Filters:

$$\text{MSE} = \mathbb{E}\{e^2[n]\} = \mathbb{E}\{(x[n] - \hat{x}[n])^2\}$$

Ziel: Finde den Filter $h[n]$, der diesen MSE minimiert.

3. Die Mathematische Herleitung (im Zeitbereich)

Wir setzen unsere Ziele in die Mathematik um.

1. Fehler aufschreiben:

$$e[n] = x[n] - \hat{x}[n] = x[n] - \sum_{k=0}^{L-1} h[k] \cdot y[n-k]$$

Hier nehmen wir an, dass unser Filter eine endliche Länge L hat (FIR-Filter).

2. Fehlerquadrat:

$$e^2[n] = \left(x[n] - \sum_{k=0}^{L-1} h[k] y[n-k] \right)^2$$

3. Erwartungswert des Fehlerquadrats (MSE):

$$\mathbb{E}\{e^2[n]\} = \mathbb{E}\left\{ \left(x[n] - \sum_{k=0}^{L-1} h[k] y[n-k] \right)^2 \right\}$$

4. Minimierung durch Ableitung:

Um das Minimum zu finden, leiten wir den MSE nach jedem Filterkoeffizienten $h[j]$ ab und setzen die Ableitung gleich Null.

$$\frac{\partial \mathbb{E}\{e^2[n]\}}{\partial h[j]} = 0 \text{ für } j = 0, 1, \dots, L-1$$

Das führt zu:

$$\mathbb{E}\left\{ 2 \cdot \left(x[n] - \sum_{k=0}^{L-1} h[k] y[n-k] \right) \cdot (-y[n-j]) \right\} = 0$$

Vereinfacht:

$$\begin{aligned}-2 \mathbb{E}\{(x[n] - \hat{x}[n]) \cdot y[n-j]\} &= 0 \\ \mathbb{E}\{e[n] \cdot y[n-j]\} &= 0\end{aligned}$$

Diese letzte Gleichung ist äußerst wichtig und wird **Orthogonalitätsprinzip** genannt. Sie besagt, dass im Optimum der Schätzfehler $e[n]$ orthogonal (unkorreliert) zu den Eingangsdaten $y[n-j]$ ist. Der Fehler enthält keine Information mehr, die aus den Eingangsdaten hätte extrahiert werden können.

Setzen wir nun $\hat{x}[n]$ wieder ein:

$$\begin{aligned}\mathbb{E}\{y[n-j] \cdot (x[n] - \sum_{k=0}^{L-1} h[k]y[n-k])\} &= 0 \\ \mathbb{E}\{y[n-j]x[n]\} - \sum_{k=0}^{L-1} h[k] \mathbb{E}\{y[n-j]y[n-k]\} &= 0\end{aligned}$$

Jetzt erkennen wir die Definitionen von Korrelationsfunktionen:

- $\mathbb{E}\{y[n-j]x[n]\}$ ist die **Kreuzkorrelation** $r_{yx}[j]$ zwischen dem verrauschten Eingang y und dem gewünschten Signal x .
- $\mathbb{E}\{y[n-j]y[n-k]\}$ ist die **Autokorrelation** $r_{yy}[j-k]$ des Eingangssignals y .

Damit erhalten wir die berühmte **Wiener-Hopf-Gleichung**:

$$r_{yx}[j] = \sum_{k=0}^{L-1} h[k] r_{yy}[j-k] \text{ für } j = 0, 1, \dots, L-1$$

4. Die Lösung: Matrixdarstellung

Die Wiener-Hopf-Gleichung stellt ein lineares Gleichungssystem mit L Unbekannten ($h[0], h[1], \dots, h[L-1]$) dar. In Matrixschreibweise sieht das so aus:

$$\begin{bmatrix} r_{yy}[0] & r_{yy}[1] & \cdots & r_{yy}[L-1] \\ r_{yy}[1] & r_{yy}[0] & \cdots & r_{yy}[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ r_{yy}[L-1] & r_{yy}[L-2] & \cdots & r_{yy}[0] \end{bmatrix} \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[L-1] \end{bmatrix} = \begin{bmatrix} r_{yx}[0] \\ r_{yx}[1] \\ \vdots \\ r_{yx}[L-1] \end{bmatrix}$$

Oder kompakt:

$$\mathbf{R}_{yy} \mathbf{h} = \mathbf{r}_{yx}$$

Die Lösung für den optimalen Filter \mathbf{h} ist damit:

$$\mathbf{h} = \mathbf{R}_{yy}^{-1} \mathbf{r}_{yx}$$

Wichtige Annahme: Wir müssen die Autokorrelation r_{yy} und die Kreuzkorrelation r_{yx} kennen! In der Praxis schätzt man sie aus Daten.

5. Die Lösung im Frequenzbereich (meist praktischer für Audio)

Wenn die Signale stationär sind und die Filterlänge sehr groß wird, ist die Berechnung im Frequenzbereich viel effizienter. Hier ist die Lösung überraschend einfach:

$$H(f) = \frac{P_{yx}(f)}{P_{yy}(f)}$$

Dabei ist:

- $H(f)$: Die Übertragungsfunktion des Wiener-Filters (die Fouriertransformierte von $h[n]$).
- $P_{yy}(f)$: Das Leistungsdichtespektrum (Power Spectral Density, PSD) des verrauschten Signals $y[n]$.
- $P_{yx}(f)$: Das Kreuzleistungsdichtespektrum zwischen $y[n]$ und $x[n]$.

Da $y[n] = x[n] + d[n]$ ist und wir oft annehmen, dass Signal $x[n]$ und Rauschen $d[n]$ unkorreliert sind ($r_{xd}[m] = 0$), gilt:

- $P_{yy}(f) = P_{xx}(f) + P_{dd}(f)$
- $P_{yx}(f) = P_{xx}(f)$

Setzen wir das ein, erhalten wir die gebräuchlichste Form des Wiener-Filters im Frequenzbereich:

$$H(f) = \frac{P_{xx}(f)}{P_{xx}(f) + P_{dd}(f)}$$

6. Intuitive Interpretation im Frequenzbereich

Diese letzte Gleichung ist physikalisch wunderbar zu verstehen:

- $P_{xx}(f)$: Die Leistung des Nutzsignals bei Frequenz f .
- $P_{dd}(f)$: Die Leistung des Rauschsignals bei Frequenz f .

Der Wiener-Filter $H(f)$ ist also ein **frequenzabhängiger Gain** (eine Art "intelligenter Equalizer").

- **Bei Frequenzen, wo das Signal viel stärker ist als das Rauschen** (hohes Signal-Rausch-Verhältnis SNR): $P_{xx}(f) \gg P_{dd}(f) \Rightarrow H(f) \approx 1$. Das Signal wird **kaum verändert**.
- **Bei Frequenzen, wo das Rauschen dominiert** (niedriges SNR): $P_{xx}(f) \ll P_{dd}(f) \Rightarrow H(f) \approx 0$. Diese Frequenzbänder werden **stark abgeschwächt** oder **ausgelöscht**.

Zusammenfassend ist das Wiener-Filter ein optimaler Kompromiss zwischen Rauschunterdrückung und Signalverzerrung.

Praxis im Audio-Bereich

In der Praxis muss man $P_{xx}(f)$ und $P_{dd}(f)$ schätzen. Eine gängige Methode ist:

1. Eine kurze Passage mit **nur Rauschen** aufnehmen (z.B. eine Pause im Sprachsignal) und daraus $P_{dd}(f)$ schätzen.
2. Während der Sprachaktivität wird $P_{yy}(f)$ gemessen.
3. Da $P_{xx}(f) = P_{yy}(f) - P_{dd}(f)$ (wegen der Unkorreliertheit), kann man $H(f)$ berechnen und auf das Signal anwenden. Oft wird dies in überlappenden Blöcken (Frames) gemacht, um sich ändernden Bedingungen gerecht zu werden (z.B. in Echtzeit-Entstörsystemen).

Die größte Herausforderung ist eine gute Schätzung der Leistungsspektren, ohne dabei Artefakte wie "Musical Noise" zu erzeugen.

Prompt: Was ist der Unterschied vom Wiener Filter zur spektralen Subtraktion? Kurze Erklärung

Reply:

Sehr kurze Erklärung:

Spektrale Subtraktion: Bereinigt = Gemessen - Rauschen
Einfache Subtraktion im Spektralbereich, führt oft zu Artefakten.

Wiener-Filter: Bereinigt = Gemessen \times (Signal/(Signal + Rauschen))
Optimaler Kompromiss zwischen Rauschreduktion und Signalverzerrung.

Etwas ausführlicher:

Spektrale Subtraktion

- **Formel:** $|X_{clean}| = |Y| - |D|$ (oft mit Begrenzung)
- **Idee:** Direktes Abziehen des geschätzten Rauschens
- **Problem:** Kann negative Werte ergeben → "Musical Noise"
- **Einfach, aber aggressiv**

Wiener-Filter

- **Formel:** $H = P_{xx} / (P_{xx} + P_{dd})$
- **Idee:** Optimaler Frequenz-Gain basierend auf SNR
- **Vorteil:** Immer zwischen 0 und 1 → natürlicherer Klang
- **Mathematisch optimal** für minimale mittlere quadratische Abweichung

Intuitive Analogie:

- **Spektrale Subtraktion:** "Alles unter Schwelle X abschneiden"
- **Wiener-Filter:** "Lautstärke-Regler der je nach Frequenz und Rauschen optimiert"

Der Wiener-Filter ist die "intelligenter" Version der spektralen Subtraktion.

[ChatGPT-Log]

Datum: 2025-10-22

Uhrzeit: 17:04

Thema/Ziel: STOI, PESQ

Prompt: Was ist "Bewerten Sie mit STOI, PESQ/ViSQOL"

Reply:

Sehr gute Frage! Das sind die **wichtigsten Metriken** zur Bewertung von Sprachqualität. Hier eine deutliche Erklärung:

 **STOI (Short-Time Objective Intelligibility)**

Was misst STOI?

Sprachverständlichkeit - Wie gut kann man die Sprache verstehen?

Wie funktioniert es?

```
python  
from pystoi import stoi
```

```
# Beispiel mit VoiceBank Daten  
  
clean, sr = librosa.load("clean.wav", sr=16000)  
  
enhanced, sr = librosa.load("enhanced.wav", sr=16000)  
  
  
stoi_score = stoi(clean, enhanced, sr, extended=False)  
  
print(f"STOI Score: {stoi_score:.3f}")
```

STOI Interpretation:

- **0.0-0.3:** Sehr schlechte Verständlichkeit ❌
- **0.3-0.5:** Schwierig zu verstehen ⚠️
- **0.5-0.7:** Akzeptable Verständlichkeit ✓
- **0.7-0.9:** Gute Verständlichkeit 👍
- **0.9-1.0:** Ausgezeichnete Verständlichkeit 🏆

Praxis-Beispiel:

- Rauschen reduziert STOI (schlechter)
- Verzerrungen reduzieren STOI (schlechter)
- Klare Sprache erhöht STOI (besser)

📞 PESQ (Perceptual Evaluation of Speech Quality)

Was misst PESQ?

Gesamt-Sprachqualität - Wie "natürlich" klingt die Sprache?

Wie funktioniert es?

```
python  
from pesq import pesq
```

```
# Achtung: Beide Signale müssen gleiche Länge haben!  
  
min_len = min(len(clean), len(enhanced))  
  
pesq_score = pesq(sr, clean[:min_len], enhanced[:min_len], 'wb')  
  
print(f"PESQ Score: {pesq_score:.3f}")
```

PESQ Interpretation:

- **-0.5 - 1.0:** Sehr schlechte Qualität ❌
- **1.0 - 2.0:** Schlechte Qualität ⚠️

- **2.0 - 3.0:** Mittlere Qualität ✓
- **3.0 - 4.0:** Gute Qualität 👍
- **4.0 - 4.5:** Sehr gute Qualität 🏆

Typische Werte für Rauschreduktion:

- Stark verrauscht: ~1.5-2.0
 - Nach spektraler Subtraktion: ~2.5-3.0
 - Professionelle Systeme: ~3.0-3.5
-

♫ ViSQOL (Virtual Speech Quality Objective Listener)

Was misst ViSQOL?

Ähnlichkeit zum Original - Wie sehr klingt es wie das Original?

Installation & Usage:

bash

```
# Installation
```

```
pip install visqol
```

python

```
import visqol
```

```
# ViSQOL ist rechenintensiv, aber sehr genau
```

```
config = visqol.Config()
```

```
config.sample_rate = sr
```

```
api = visqol.VisqolApi()
```

```
api.Create(config)
```

```
# Berechnung
```

```
visqol_score = api.Measure(clean, enhanced)
```

```
print(f"ViSQOL Score: {visqol_score:.3f}")
```

ViSQOL Interpretation:

- **1.0-3.0:** Schlechte Ähnlichkeit ✗
- **3.0-4.0:** Mittlere Ähnlichkeit ⚠
- **4.0-4.5:** Gute Ähnlichkeit ✓
- **4.5-5.0:** Sehr gute Ähnlichkeit 🏆

[DeepSeek-Log]

Datum: 2025-11-15

Uhrzeit: 14:03

Thema/Ziel: STOI implementierung

Prompt: Das ist mein Code (Anhang Spectral Subtraction), wie füge ich stoi dem projekt hinzu?

Reply:

So fügen Sie STOI zu Ihrem Projekt hinzu:

1. STOI installieren

bash

```
pip install pystoi
```

2. STOI-Evaluation in Ihren Code integrieren

Einfache Integration:

python

```
import numpy as np
```

```
from pystoi import stoi
```

```
import librosa
```

```
import soundfile as sf
```

```
def evaluate_with_stoi(clean_audio, enhanced_audio, sr, method_name=""):
```

```
    """
```

Evaluierung mit STOI - Sprachverständlichkeit-Metrik

```
    """
```

```
try:
```

```
    # STOI berechnen (Werte zwischen 0-1, höher = besser)
```

```
    stoi_score = stoi(clean_audio, enhanced_audio, sr, extended=False)
```

```
    print(f"STOI Evaluation {method_name}:")
```

```
    print(f" STOI Score: {stoi_score:.3f}")
```

```
    print(f" Interpretation: {'Hervorragend' if stoi_score > 0.9 else 'Gut' if stoi_score > 0.75 else 'Mäßig' if stoi_score > 0.6 else 'Schlecht'}")
```

```
return stoi_score
```

```
except Exception as e:
```

```
    print(f"STOI Berechnung fehlgeschlagen: {e}")
```

```

    return None

Erweiterte Evaluierungsfunktion:

python

def comprehensive_evaluation(clean_audio, noisy_audio, enhanced_audio, sr, method_name=""):

    """
    Umfassende Evaluierung mit STOI und anderen Metriken
    """

    results = {}

    # STOI - Sprachverständlichkeit
    try:
        results['stoi'] = stoi(clean_audio, enhanced_audio, sr, extended=False)
    except Exception as e:
        print(f"STOI Fehler: {e}")
        results['stoi'] = 0.0

    # SNR Verbesserung
    original_snr = calculate_snr(clean_audio, noisy_audio - clean_audio)
    enhanced_snr = calculate_snr(clean_audio, enhanced_audio - clean_audio)
    results['snr_improvement'] = enhanced_snr - original_snr

    # Einfache Qualitätsmetrik (falls PESQ nicht verfügbar)
    results['quality_score'] = calculate_simple_quality(clean_audio, enhanced_audio)

    # Ergebnisse ausgeben
    print(f"\n==== EVALUATION: {method_name} ====")
    print(f"STOI (Verständlichkeit): {results['stoi']:.3f}")
    print(f"SNR Verbesserung: {results['snr_improvement']:.2f} dB")
    print(f"Qualitäts-Score: {results['quality_score']:.3f}")

    return results

def calculate_snr(signal, noise):
    """Berechnet Signal-zu-Rausch Verhältnis"""

```

```

signal_power = np.mean(signal**2)

noise_power = np.mean(noise**2)

return 10 * np.log10(signal_power / (noise_power + 1e-10))

```

```

def calculate_simple_quality(clean, enhanced):

    """Einfache Qualitätsmetrik als Fallback"""

    correlation = np.corrcoef(clean, enhanced)[0,1]

    return max(0, correlation) # Normiert auf 0-1

```

3. Komplette Hauptfunktion mit STOI

```

python

def main():

    print("Spectral Subtraction mit STOI Evaluation")

```

```

try:

    # Audio laden

    noisy, sr = librosa.load("p232_014_noiseWithMusic.wav", sr=16000)

```

```
# Versuche clean Audio zu laden für echte STOI Evaluation
```

```

try:

    clean, sr_clean = librosa.load("p232_014_clean.wav", sr=16000)

    if len(clean) != len(noisy):

        print("Achtung: Clean und Noisy haben unterschiedliche Längen")

        clean = clean[:len(noisy)] # Anpassen

```

```

except:

    print("Clean Referenz nicht gefunden - verwende noisy als Referenz")

    clean = None

```

```

# Spectral Subtraction anwenden

enhanced = simple_spectral_subtraction(noisy, sr)

```

```

# STOI Evaluation

if clean is not None:

    stoi_score = evaluate_with_stoi(clean, enhanced, sr, "Spectral Subtraction")

```

```

# Parameter-Optimierung

best_params, best_stoi = compare_parameters_with_stoi(clean, noisy, sr)

else:

    # Fallback: STOI zwischen noisy und enhanced

    stoi_score = stoi(noisy, enhanced, sr, extended=False)

    print(f"STOI Verbesserung (noisy→enhanced): {stoi_score:.3f}")



# Ergebnisse speichern

sf.write("enhanced_with_stoi_eval.wav", enhanced, sr)

print("Evaluation abgeschlossen!")


except Exception as e:

    print(f"Fehler: {e}")

```

STOI Interpretation:

- **> 0.9:** Hervorragende Verständlichkeit
- **0.75-0.9:** Gute Verständlichkeit
- **0.6-0.75:** Mäßige Verständlichkeit
- **< 0.6:** Schlechte Verständlichkeit

Jetzt haben Sie eine **professionelle STOI-Evaluierung** in Ihrem Projekt!

Prompt: ist stoi immer ein vergleich mit einem anderen signal?

Reply:

Ja, genau! STOI ist immer ein Vergleich zwischen zwei Signalen.

Wie STOI funktioniert:

STOI (Short-Time Objective Intelligibility) vergleicht zwei Audiosignale miteinander:

- Referenzsignal (meist das "saubere"/"ideale" Signal)
- Testsignal (das zu bewertende Signal)

Typische STOI-Vergleiche:

1. Qualitätsbewertung mit Clean-Referenz

python

```
# Vergleich: Sauberes Original vs. Verarbeitetes Signal
```

```
stoi_score = stoi(clean_signal, enhanced_signal, sr)
```

- STOI ≈ 1.0: Enhanced klingt wie das saubere Original → perfekt
- STOI ≈ 0.0: Enhanced ist völlig anders → schlecht

2. Veränderungsmessung

python

```
# Vergleich: Noisy Original vs. Enhanced Version  
stoi_score = stoi(noisy_signal, enhanced_signal, sr)
```

- STOI ≈ 1.0: Kaum Veränderung → wenig Rauschreduktion
- STOI ≈ 0.7: Deutliche Veränderung → starke Rauschreduktion (mögliche Artefakte)

Für Ihre Aufgabe bedeutet das:

Ohne Clean-Referenz:

python

```
# Das können Sie JETZT machen:
```

```
stoi_noisy_vs_enhanced = stoi(noisy_audio, enhanced_audio, sr)
```

- Misst wie stark sich das Signal verändert hat
- Aber: Keine Aussage über absolute Qualität

Mit Clean-Referenz (ideal):

python

```
# Das wäre optimal:
```

```
stoi_clean_vs_noisy = stoi(clean_audio, noisy_audio, sr) # Original-Qualität  
stoi_clean_vs_enhanced = stoi(clean_audio, enhanced_audio, sr) # Verbesserte Qualität
```

- Misst absolute Sprachverständlichkeit

Praktische Lösung für Sie:

python

```
def comprehensive_evaluation(noisy_audio, enhanced_audio, sr, clean_audio=None):
```

"""

Umfassende Evaluation mit allen verfügbaren Signalen

"""

```
print("\n" + "="*60)
```

```
print("COMPREHENSIVE STOI EVALUATION")
```

```
print("="*60)
```

```
# 1. Immer möglich: Vergleich noisy vs enhanced
```

```
stoi_change = stoi(noisy_audio, enhanced_audio, sr)
```

```
print(f"Veränderung (noisy → enhanced): {stoi_change:.4f}")
```

```
print(f" - Interpretation: {'Wenig Veränderung' if stoi_change > 0.9 else 'Mäßige Veränderung' if stoi_change > 0.7  
else 'Starke Veränderung'}")
```

```

# 2. Falls Clean-Referenz verfügbar: Absolute Qualität

if clean_audio is not None:

    stoi_noisy_quality = stoi(clean_audio, noisy_audio, sr)

    stoi_enhanced_quality = stoi(clean_audio, enhanced_audio, sr)

    print(f"\nAbsolute Qualität (vs clean reference):")

    print(f" Noisy: {stoi_noisy_quality:.4f}")

    print(f" Enhanced: {stoi_enhanced_quality:.4f}")

    print(f" Verbesserung: {stoi_enhanced_quality - stoi_noisy_quality:+.4f}")

print("=*60)

```

Zusammenfassung:

- STOI braucht immer 2 Signale zum Vergleichen
- Ohne Clean-Referenz: Sie können nur die Veränderung messen
- Mit Clean-Referenz: Sie können absolute Qualität bewerten

Für Ihren Code jetzt: Verwenden Sie den Vergleich noisy vs enhanced - das zeigt ob Ihr Algorithmus das Signal signifikant verändert!

Prompt: zu stoi soll noch PESQ/ViSQOL umgesetzt werden. Was ist das?

Reply:

PESQ und ViSQOL sind weitere wichtige Metriken für Sprachqualität - hier die Erklärung:

Was ist PESQ?

- Perceptual Evaluation of Speech Quality
- Standardisiert als ITU-T P.862
- Bewertet Sprachqualität auf einer Skala von -0.5 bis 4.5
- Wird verwendet für: Telefonie, Sprachcodecs, Rauschunterdrückung
- Misst: Störungen, Verzerrungen, Rauschen

Was ist ViSQOL?

- Virtual Speech Quality Objective Listener
- Von Google entwickelt als verbesserte Version von PESQ
- Besser für moderne Codecs und starke Kompression
- Skala: Ähnlich wie PESQ, aber genauer für nicht-lineare Verzerrungen

Unterschied zu STOI:

Metrik	Was wird gemessen?	Skala	Stärken
STOI	Sprachverständlichkeit	0-1	Wie gut Sprache verstanden wird
PESQ	Sprachqualität	-0.5 bis 4.5	Wie natürlich/angenehm es klingt
ViSQOL	Sprachqualität (modern)	-0.5 bis 4.5	Besser für moderne Audio-Verarbeitung

Installation:

bash

pip install pesq

ViSQOL ist komplexer zu installieren - wir beginnen mit PESQ

Jetzt erhalten Sie ein komplettes Bild: STOI für Verständlichkeit + PESQ für Klangqualität!

Prompt: warum sind die Ergebnisse der optimierten Werte so unterschiedlich?

OPTIMAL PARAMETERS FOR STOI (Speech Intelligibility):

Alpha: 2.0, Beta: 0.05, N_FFT: 1024

STOI: 0.9197

OPTIMAL PARAMETERS FOR PESQ (Speech Quality):

Alpha: 5.0, Beta: 0.01, N_FFT: 1024

PESQ: 1.84

Reply:

Das ist **vollkommen normal** und zeigt genau den **Trade-off zwischen Sprachverständlichkeit und Klangqualität!**

Warum die Parameter so unterschiedlich sind:

STOI-Optimierung (Alpha: 2.0, Beta: 0.05)

- **Konservativer Ansatz:** Weniger aggressive Rauschunterdrückung
- **Höheres Beta (0.05):** Bewahrt mehr vom Originalsignal (verhindert "musical noise")
- **Mittleres Alpha (2.0):** Ausgewogene Rauschreduktion
- **Ergebnis:** Gute Sprachverständlichkeit, aber mehr Restrauschen

PESQ-Optimierung (Alpha: 5.0, Beta: 0.01)

- **Aggressiver Ansatz:** Starke Rauschunterdrückung
- **Niedrigeres Beta (0.01):** Erlaubt stärkere Signalveränderung
- **Hohes Alpha (5.0):** Sehr aggressive Rauschsubtraktion
- **Ergebnis:** Weniger Rauschen, aber mögliche Sprachverzerrungen

Typische Trade-offs bei Spectral Subtraction:

Parameter STOI bevorzugt PESQ bevorzugt Effekt

Alpha	1.5-3.0	3.0-8.0	Höheres Alpha = aggressivere Rauschreduktion
Beta	0.02-0.1	0.001-0.02	Niedrigeres Beta = mehr Signalveränderung

Interpretation Ihrer Ergebnisse:

- **STOI 0.9197:** Hervorragende Sprachverständlichkeit - Sprache ist gut verständlich
- **PESQ 1.84:** Mäßige Klangqualität - klingt verzerrt/unnatürlich, aber weniger Rauschen

Das bedeutet praktisch:

- **Für Telefonie/Spracherkennung:** STOI-Parameter verwenden (Sprache verständlich)
- **Für Musik/Hörgenuss:** PESQ-Parameter verwenden (weniger Rauschen)
- **Für Kompromiss:** Parameter dazwischen wählen (z.B. Alpha: 3.0, Beta: 0.02)

Dies zeigt genau den klassischen Trade-off in der Audio-Verarbeitung:

- **Entweder** gute Sprachverständlichkeit **oder** angenehmer Klang
- **Selten** beides gleichzeitig optimal möglich

Das ist ein **erwartetes und normales Ergebnis** Ihrer Optimierung!

Prompt: was macht mmse genau?

Reply:

Das ist interessant! Lass mich erklären, was MMSE macht:

Was macht MMSE genau?

MMSE (Minimum Mean Square Error) ist ein statistischer Ansatz zur Rauschunterdrückung:

Kernidee:

- **Mathematisch optimiert:** Minimiert den mittleren quadratischen Fehler zwischen sauberem und geschätztem Signal
- **SNR-basiert:** Berechnet für jede Frequenz eine optimale Gewichtung basierend auf Signal-zu-Rausch-Verhältnis
- **Weicher Übergang:** Erzeugt weniger "musical noise" als Spectral Subtraction

Unterschied zu Spectral Subtraction:

Feature Spectral Subtraction MMSE

Ansatz Direkte Subtraktion Statistische Optimierung

Artefakte Mehr "musical noise" Weniger "musical noise"

Berechnung Einfacher Komplexer

Qualität Oft aggressiver Oft natürlicher

[Gemini-Log]

Datum: 2026-01-02

Uhrzeit: 17:34

Thema/Ziel: Code Review

Prompt: Hier ist mein Code. Schau mal, ob du Fehler findest. (Anhang Code) Sind die Algorithmen korrekt umgesetzt?

Reply:

Das ist eine solide Grundlage! Du hast die klassischen Algorithmen (Spectral Subtraction nach Berouti/Boll und Wiener Filter mit Decision-Directed Approach) strukturell gut umgesetzt.

Allerdings gibt es **drei spezifische Probleme** (zwei logische/algorithemische und einen potenziellen Absturz-Bug), die du beheben solltest, damit die Ergebnisse korrekt sind und der Code robust läuft.

Hier ist die Analyse deiner Algorithmen mit Verbesserungsvorschlägen:

1. Spectral Subtraction (spectral_subtraction)

Das Problem: Der "Spectral Floor" (Berouti-Methode) ist wahrscheinlich falsch implementiert. Du rechnest: `np.maximum(power_clean, beta * power_noisy)`. Der Standard-Algorithmus setzt das Minimum (den Floor) normalerweise in Relation zum **Rauschen** (`power_noise`), nicht zum verrauschten Signal (`power_noisy`).

- **Warum das problematisch ist:** Wenn du `beta * power_noisy` nimmst, ist dein "Floor" bei lauten Sprachanteilen plötzlich riesig (weil `power_noisy` dort hoch ist). Das kann dazu führen, dass du in Sprachpausen den Floor niedrig hast, aber während der Sprache den Floor künstlich anhebst und so Verzerrungen einbaust.
- **Korrektur:** Der Floor sollte `beta * power_noise` sein. Damit bleibt das Rauschen konstant niedrig und "musikalisch" angenehmer.

2. Wiener Filter (wiener_filter)

Hier gibt es ein kritisches Problem mit den Dimensionen und der Stabilität.

Problem A: Dimension Mismatch (Potential Crash) In deiner Schleife: `gamma = power_noisy[:, t:t+1] / power_noise`. Falls deine `noise_estimation` Funktion eine **zeitvariable** Rauschschätzung zurückgibt (also eine Matrix mit Shape [Frequenzen, Zeit]), wird dieser Code abstürzen oder falsch rechnen.

- `power_noisy[:, t:t+1]` hat Shape (F, 1).
- `power_noise` hat Shape (F, T). Das lässt sich nicht direkt dividieren. Du musst auch vom Noise den aktuellen Frame t nehmen.

Problem B: Division durch Null Du hast zwar ein `eps` definiert, nutzt es aber nicht bei der Berechnung von `gamma`. Wenn `power_noise` an einer Stelle 0 ist (unwahrscheinlich, aber möglich), gibt es NaN oder inf.

Möchtest du, dass ich mir die `noise_estimation` Funktion auch ansehe, falls du sie hast? Davon hängt die Qualität oft am stärksten ab.

Prompt: Ich schicke dir noch weitere Code Dateien (Anhang Code)

Reply:

Das Batch-Processing-Skript ist **strukturell sehr gut aufgebaut**. Du hast an fast alles gedacht: Argument-Parsing, Resume-Funktionalität, Logging, JSON-Export und Sampling.

Es gibt jedoch **drei kritische Bereiche**, die wahrscheinlich zu Abstürzen oder logischen Fehlern führen werden, wenn du das Skript auf viele Dateien loslässt.

Hier sind die Probleme und die korrigierten Code-Abschnitte.

Die 3 Hauptprobleme

1. **JSON Serialisierungs-Absturz (Kritisch):** Die Parameter in param_ranges sind oft NumPy-Datentypen (z.B. np.float64 oder np.int32 aus np.linspace). Das Standard json Modul in Python kann diese **nicht speichern**. Das Skript wird ganz am Ende beim Speichern der all_results.json mit einem TypeError abstürzen, und deine stundenlange Berechnung ist weg.
2. **Resume-Logik ist zu grob:** Deine Funktion get_processed_stems markiert eine Datei als "fertig", sobald **irgendein** Algorithmus (z.B. nur Spectral Subtraction) Ergebnisse liefert hat. *Szenario:* Du startest das Skript, es rechnet "SS" und "Wiener" für Datei A, stürzt dann bei "OMLSA" ab. *Folge:* Beim Neustart (--) resume) sieht das Skript: "Datei A hat Ergebnisse", und überspringt sie komplett. "OMLSA" wird für Datei A niemals berechnet.
3. **PESQ Längen-Crash in der Schleife:** In optimize_parameters rufst du calculate_pesq auf. Manche Algorithmen (durch Rundung bei STFT/ISTFT) geben ein Array zurück, das 1-2 Samples kürzer oder länger ist als die Referenz. PESQ stürzt sofort ab, wenn die Längen nicht exakt gleich sind (je nach Wrapper-Implementierung).

Deine Rauschschätzung ist architektonisch **hervorragend** umgesetzt. Die Verwendung einer abstrakten Basisklasse (NoiseEstimator) und einer Factory-Methode macht das System extrem modular und perfekt für deine Optimierungsschleife geeignet.

Der Code ist logisch korrekt und lässt sich nahtlos in die spectral_subtraction und den wiener_filter integrieren, die wir zuvor besprochen haben.

Hier sind meine Anmerkungen zur Korrektheit und zwei wichtige Optimierungsvorschläge:

1. Analyse der Schätzer

- **PercentileNoiseEstimator:** Sehr robust. Dass du die Log-Energie für die Sortierung nutzt, ist klug, da dies Ausreißer in der Leistung (Power) dämpft. Die adaptive Logik für kurze Signale verhindert, dass der Schätzer bei kurzen Sprachaufnahmen "abstürzt".
- **MinTrackingNoiseEstimator:** Die Idee ist gut, aber die Implementierung von _sliding_minimum mit einer Python-Schleife wird bei langen Dateien (z. B. 30 Sekunden Audio) sehr **langsam**. Da deine Optimierungsschleife den Algorithmus hunderte Male aufruft, ist das ein Flaschenhals.
- **TrueNoiseEstimator:** Mathematisch korrekt (Power_Noisy - Power_Clean). Dass du den Mittelwert über die Zeit bildest, liefert ein stabiles, statisches Profil.

Möchtest du als Nächstes den **MMSE (Minimum Mean Square Error)** Algorithmus implementieren, oder sollen wir uns die parameter_ranges für die Optimierung anschauen?

[Gemini-Log]

Datum: 2026-01-04

Uhrzeit: 13:30

Thema/Ziel: Code Review

Prompt: schau dir nun den code von mmse an. ist der korrekt? (Anhang Code)

Reply:

Dein Code für den **MMSE-STSA (Minimum Mean Square Error Short-Time Spectral Amplitude Estimator)** nach Ephraim und Malah (1984) ist mathematisch **absolut präzise** umgesetzt.

Du hast die klassische Gain-Funktion, die auf modifizierten Besselfunktionen basiert, korrekt implementiert. Besonders positiv hervorzuheben ist, dass du v auf 80.0 begrenzt hast, um numerische Overflows bei der Berechnung der Besselfunktionen zu vermeiden – das ist ein häufiger Stolperstein.

Hier ist die detaillierte Analyse und ein paar Tipps für die Robustheit:

1. Mathematische Korrektheit

Die von dir implementierte Formel entspricht exakt dem Standardwerk:

Dabei ist:

- γ : A-priori SNR
- δ : A-posteriori SNR

2. Mögliche Probleme & Optimierungen

A. Stabilität der Besselfunktionen (Vorschlag: ive)

Obwohl du v auf 80 begrenzt hast, gibt es eine noch stabilere Methode. SciPy bietet `ive(n, z)`, was entspricht. Da die MMSE-Formel sowieso den Term enthält, kannst du beide zusammenfassen. Das verhindert "Inf * 0" Fehler komplett, selbst ohne hartes Clipping von v .

Optimierter Code-Ausschnitt:

Python

```
from scipy.special import ive

# Statt i0/i1 und exp separat:
# exp_term = np.exp(-0.5 * v)
# i0_term = i0(0.5 * v)
# i1_term = i1(0.5 * v)
# bessel_part = (1.0 + v) * i0_term + v * i1_term
# gain = (np.sqrt(np.pi) / 2.0) * (np.sqrt(v) * exp_term * bessel_part) / (gamma + eps)
```

```
# Nutze ive für maximale Stabilität:
```

```
i0_scaled = ive(0, 0.5 * v)
i1_scaled = ive(1, 0.5 * v)
bessel_part_scaled = (1.0 + v) * i0_scaled + v * i1_scaled
gain = (np.sqrt(np.pi) / 2.0) * (np.sqrt(v) * bessel_part_scaled) / (gamma + eps)
```

B. Das "Noise-Slice" Problem

Wie schon beim Wiener Filter erwähnt: Falls dein noise_estimation Tool eine Matrix zurückgibt (zeitvariables Rauschen), wird $\gamma = \text{power_noisy}[:, t:t+1] / \text{power_noise}$ fehlschlagen.

Fix:

Python

```
# Prüfen, ob Noise 2D ist  
is_noise_2d = (power_noise.ndim == 2 and power_noise.shape[1] > 1)
```

```
for t in range(num_frames):  
  
    current_noise = power_noise[:, t:t+1] if is_noise_2d else power_noise  
  
    gamma = power_noisy[:, t:t+1] / (current_noise + eps)  
  
    # ... ksi Berechnung entsprechend mit current_noise ...
```

C. Initialisierung von prev_gain

Du hast `prev_gain = None`. In der Schleife wird beim ersten Durchlauf ($t == 0$) `ksi` korrekt berechnet, aber `prev_gain` wird erst am Ende von Frame 0 gesetzt. Das ist okay, da du `if t == 0` abfragst. Sauberer für die Performance (Vermeidung der `if`-Abfrage in jedem der tausenden Schleifendurchläufe) wäre eine Vor-Initialisierung.

Zusammenfassung

Der Algorithmus ist **top**. MMSE-STSA ist bekannt dafür, das "Musical Noise" (typisch für Spectral Subtraction) deutlich besser zu unterdrücken, da die Gain-Funktion weniger aggressiv bei niedrigen SNRs abschneidet.

Soll ich dir zeigen, wie du den Wiener Filter und MMSE so kombinierst, dass du sie in deiner Batch-Optimierung direkt gegeneinander antreten lassen kannst? (Du hast das im Batch-Skript zwar schon drin, aber man könnte eine gemeinsame Metrik-Tabelle für den Vergleich erstellen).

[ChatGPT-Log]

Datum: 2026-01-07

Uhrzeit: 13:47

Thema/Ziel: OM-LSA

Prompt: was ist omlsa im Audio Kontext?

Reply:

OM-LSA steht für “Optimally Modified Log-Spectral Amplitude” (oft auch **OMLSA** geschrieben). Es ist im Kern eine **Log-MMSE/LSA-Sprachverbesserung**, die explizit berücksichtigt, dass Sprache in einem Zeit-Frequenz-Bin **nicht sicher vorhanden** ist (“signal/speech presence uncertainty”).

Idee in einem Satz

Statt überall den Log-MMSE-Gain anzuwenden, mischt OM-LSA den Gain **soft** abhängig von der **Sprachpräsenz-Wahrscheinlichkeit**: wenn vermutlich **keine** Sprache da ist, wird stärker Richtung **Gain-Floor** gedämpft; wenn Sprache da ist, bleibt man nahe am Log-MMSE-Gain.

Was macht OM-LSA genau?

1) Zwei Hypothesen pro Bin

Für jedes (m, k) werden zwei Fälle betrachtet:

- \mathcal{H}_0 : **Sprache abwesend**, nur Noise

- \mathcal{H}_1 : Sprache präsent (Speech+Noise)

Darauf wird eine **a-priori Sprachabwesenheits-Wahrscheinlichkeit** $q = P(\mathcal{H}_0)$ und daraus via Bayes eine **a-posteriori Sprachpräsenz-Wahrscheinlichkeit** $p = P(\mathcal{H}_1 | Y)$ berechnet.

2) Gain als gewichtetes geometrisches Mittel (der “optimally modified” Teil)

Cohen zeigt, dass der resultierende OM-LSA-Gain als **gewichtetes geometrisches Mittel** der hypothetischen Gains formuliert werden kann:

- bei \mathcal{H}_1 : der normale **LSA / Log-MMSE-Gain**
- bei \mathcal{H}_0 : ein **kleiner Minimalgain** G_{\min} (“noise naturalness” / Gain-Floor)

Das führt direkt zur typischen Form

$$G(m, k) = G_{\text{LSA}}(m, k)^{p(m, k)} G_{\min}.$$

Genau diese geometrische Mischung und die Motivation des **Gain-Floors** stehen im Cohen-Paper und werden im Loizou-Buch als “optimally modified” log-spectral amplitude estimator diskutiert.

3) Warum hilft das?

Cohen berichtet, dass OM-LSA gegenüber LSA/STSA/MM-LSA **besser** unterdrückt, besonders bei **niedrigen SNRs** und **nichtstationärem Lärm**, und dabei **musical noise** reduziert, weil Noise-Bins gezielter erkannt und stärker gedämpft werden.

Wenn du möchtest, kann ich dir auch:

- eine saubere formale Definition für deine Thesis formulieren
- oder eine kompakte Grundlagenbeschreibung mit Zitierform

Sag einfach Bescheid

Prompt: Was ist der Unterschied zwischen OMLSA und Log MMSE?

Reply:

Der Unterschied ist im Kern:

Log-MMSE (LSA-MMSE) schätzt die Sprach-Amplitude **unter der Annahme „Sprache ist da“**.

OM-LSA erweitert Log-MMSE um **Signal-/Sprachpräsenz-Unsicherheit** (SPP/SAP) und mischt den Gain entsprechend.

Log-MMSE (LSA-MMSE): „Sprache ist vorhanden“

- Optimiert den Fehler der **logarithmischen spektralen Amplitude** (daher „log“).
- Berechnet einen Gain $G_{\text{LSA}}(\xi, \gamma)$ (typisch über Exponentialintegral $E_1(\cdot)$).
- Keine explizite Wahrscheinlichkeit, ob in einem Zeit-Frequenz-Bin überhaupt Sprache ist.

OM-LSA: Log-MMSE + Sprachpräsenzwahrscheinlichkeit (SPP/SAP)

OM-LSA (Cohen, 2002) macht genau das:

- Modelliert zwei Hypothesen pro TF-Bin:
 H_0 (nur Rauschen) vs. H_1 (Sprache+Rauschen)

- Schätzt **Speech Presence Probability** $p(H_1 | Y)$ (bzw. Speech Absence Probability)
- Kombiniert Gains als **gewichtetes geometrisches Mittel** aus „Sprache da“-Gain und „Sprache weg“-Gain (mit Floor), um **musical noise zu reduzieren** und schwache Sprachanteile zu schonen.

Loizou beschreibt diese Idee als „log-MMSE mit speech-presence uncertainty“ (Kap. 7.13.3) und zeigt, dass der „optimally modified“ Gain dann **multiplikativ** wird (statt einer unpraktischen nicht-multiplikativen Form).

Praktische Auswirkungen (hörbar)

- **Log-MMSE:** oft schon „smooth“/wenig musikalisch, aber kann bei starkem Rauschen in noise-only TF-Bins trotzdem artefaktig werden.
 - **OM-LSA:** dämpft in TF-Bins, die wahrscheinlich **nur Rauschen** sind, stärker (bis zum Floor) und ist daher meist **robuster bei nichtstationärem Rauschen und weniger musical noise**.
-

Bonus: Bezug zu deinem Code

Dein advanced_mmse macht explizit:

- Log-MMSE Gain via $E_1(\expn(1,v))$
- SPP aus q
- geometrische Mischung mit gain_floor:

$$G = G_{\text{LSA}}^p \cdot G_{\text{floor}}^{(1-p)}$$

Das ist genau der OM-LSA-Mechanismus.

[ChatGPT-Log]

Datum: 2026-02-02

Uhrzeit: 13:30

Thema/Ziel: Text Review

Prompt: Schau dir meinen Text an. Ist der inhaltlich so korrekt? (Anhang Latex Datei Grundlagen)

Reply:

Hier sind die wichtigsten Korrekturen (inhaltlich + formal), damit dein Abschnitt sauber mit den Quellen übereinstimmt:

- **Begriffe/Notation präzisieren:** In den Originalquellen wird i. d. R. mit **STFT-Koeffizienten** $Y(m, k)$ (komplex), deren **Betrag** $R(m, k) = |Y(m, k)|$ und den **Varianzen** $\lambda_x(k), \lambda_d(k)$ gearbeitet; daraus folgen γ und ξ .
- **Gain-Formel leicht korrigieren:** Deine Gain-Gleichung ist praktisch richtig, aber die Standardform ist klarer als $G(\xi, \gamma)$ und wird als $\hat{G} = G(\xi, \gamma) R$ geschrieben.
- **Aussage zur Phase schärfen:** Beim MMSE-STSA wird **typischerweise die verrauschte Phase** verwendet; genau das wird in Ephraim & Malah (1984) explizit begründet (der „optimale“ komplexe Exponential-Schätzer hat sonst i. A. keinen Einheitsbetrag und würde die Amplitudenschätzung wieder „kaputt machen“).
- **„Rayleigh-Verteilung“ sauber formulieren:** Rayleigh gilt für den Betrag einer **nullmittelwertigen komplexen Gaußgröße** (also für $A = |X|$ unter dem Modell), nicht „die Amplituden des Sprachsignals generell“.

- **Musical-noise-Claim entdramatisieren:** MMSE-STSA reduziert typischerweise Musical Noise gegenüber spektraler Subtraktion, aber „signifikant“ klingt ohne Kontext sehr stark. In Loizou wird auch betont, dass je nach Schätzer/Parametrisierung **Restgeräusch** weiterhin hörbar sein kann.

[ChatGPT-Log]

Datum: 2026-02-03

Uhrzeit: 12:53

Thema/Ziel: Auswertung

Prompt: Wie könnte ich meine Ergebnisse in der Arbeit gut vergleichen? (Anhang Daten)

Ich hab getestet: (alles mit sprache im vordergrund) und alles einmal in der Variante mit true noise oder ohne true noise

- rauschen mit männlicher stimme
- rauschen mit weiblicher stimme
- musik mit männlicher stimme
- musik mit weiblicher stimme
- kombinationen aus rauschen, menschen, musik
- menschen im Hintergrund/viel reden im hintergrund mit männlicher stimme
- menschen im Hintergrund/viel reden im hintergrund mit weiblicher stimme

Reply:

Das ist eine **sehr gute und ungewöhnlich saubere Datenbasis** – damit kannst du in der Arbeit wirklich überzeugend vergleichen 👍

Ich würde dir raten, den Vergleich **mehrstufig und klar getrennt** aufzubauen, statt alles auf einmal zu zeigen. Unten ein **konkreter, gut prüfbarer Vergleichs-Fahrplan**, der auch für Gutachter:innen gut lesbar ist.

1. Zuerst: Vergleichslogik klar definieren (Methodenteil!)

Bevor du Ergebnisse zeigst, solltest du explizit festlegen:

1.1 Was wird verglichen?

- **Algorithmen:**
Spectral Subtraction, Wiener, OMLSA
- **Optimierungsziel:**
 - STOI-optimiert
 - PESQ-optimiert
 - Balanced (z. B. normierte Kombination)
- **Noise-Wissen:**
 - mit *true noise*
 - ohne *true noise*
- **Szenarien:**
 - Rauschen / Musik / Menschen / Mischformen

- männliche vs. weibliche Stimme

→ Wichtig: **Nicht alles gleichzeitig vergleichen**, sondern **Dimension für Dimension**.

2. Baseline: Wie schlecht ist das Noisy-Signal?

Das hast du perfekt vorbereitet 

Das sollte **immer** der erste Vergleich sein.

Tabelle (oder kurzer Absatz):

Maß Noisy Mean

STOI ≈ 0.85

PESQ ≈ 1.18

SNR ≈ 0.36 dB

👉 Interpretation (wichtig für Text):

- Sehr geringe Sprachqualität (PESQ)
 - Verständlichkeit noch moderat (STOI)
 - Realistische, anspruchsvolle Ausgangslage
-

3. Algorithmus-Vergleich bei *Balanced Optimum* (Kernvergleich)

Das ist **der wichtigste Vergleich** für die Arbeit.

3.1 Mittelwerte (global)

Algorithmus STOI PESQ SNR

Spectral Subtraction **0.957 3.09 15.0 dB**

Wiener 0.924 2.87 14.9 dB

OMLSA 0.928 2.84 14.6 dB

3.2 Interpretation (das ist Gold für den Text)

- **Spectral Subtraction überrascht:**
 - Höchste PESQ **und** höchste STOI
 - Sehr hohe SNR-Verbesserung
 - → aber: Risiko von Artefakten (*musical noise*) → später diskutieren!
- **Wiener & OMLSA:**
 - Sehr ähnliche Performance
 - Stabil, wenig Ausreißer
 - Klassisch „robuste“ Verfahren

4. STOI- vs. PESQ-Optimierung (Zielkonflikt zeigen!)

Das ist ein **sehr starker Punkt deiner Arbeit**, den viele Arbeiten nicht sauber zeigen.

4.1 Beispiel MMSE (typisch!)

Optimierung STOI PESQ

STOI-opt **0.843** 1.27

PESQ-opt 0.819 **1.61**

➡ Klassischer Trade-off:

- PESQ-Optimierung verschlechtert STOI
- STOI-Optimierung bringt kaum Qualitätsgewinn

👉 Das kannst du für **alle Algorithmen** kurz tabellarisch oder grafisch zeigen
(z. B. Scatter-Plot STOI vs. PESQ)

5. Szenarienvergleich (sehr wichtig!)

Hier trennst du deine Arbeit von „reinen Benchmark-Papern“.

5.1 Gruppieren die Tests sinnvoll

Ich würde **nicht jede Kombination einzeln zeigen**, sondern **Clustern**:

A) Stationäres Rauschen

- Rauschen + männlich
- Rauschen + weiblich

B) Strukturierter Störer

- Musik + Sprache
- Menschen im Hintergrund

C) Hochdynamisch / realistisch

- Mischungen aus Musik + Menschen + Rauschen

5.2 Erwartbare & erklärbare Trends (die du prüfen kannst)

• Spectral Subtraction:

- sehr gut bei stationärem Rauschen
- schwächer bei Musik & Stimmen (Artefakte!)

• Wiener / OMLSA:

- stabil bei Hintergrundsprechern
- weniger aggressiv → geringere PESQ-Spitzen

👉 Hier reichen **Mittelwerte pro Szenario + kurzer Text**
(keine 20 Tabellen!)

6. True Noise vs. geschätztes Noise (separates Unterkapitel!)

Das ist **methodisch extrem wertvoll**.

- *true noise*
- ohne *true noise*

👉 sehr gut für Diskussion

7. Einzelfallanalyse (z. B. p232_138)

Die JSON-Beispiele sind perfekt für:

7.1 Tabelle: ein Signal, alle Algorithmen

Alg	STOI_bal	PESQ_bal	SNR_bal
SS	0.883	3.43	14.2
Wiener	0.815	2.83	13.7
OMLSA	0.823	2.88	13.5

👉 Danach:

- 1–2 Spektrogramme
 - 1 Hörbeispiel (wenn erlaubt)
-

8. Empfohlene Struktur im Ergebniskapitel

So liest sich das für Gutachter sehr angenehm:

1. Ausgangssignal (Noisy)
 2. Globaler Algorithmusvergleich (Balanced)
 3. Optimierungsziele (STOI vs. PESQ)
 4. Szenarienanalyse
 5. Einfluss von true noise
 6. Einzelfallanalyse
 7. Zwischenfazit
-

9. Wenn du willst

Ich kann dir als Nächstes:

- eine **konkrete LaTeX-Tabellenstruktur**
- Beispiel-Formulierungen für **Ergebnis- und Diskussionsteil**

Sag einfach, was du brauchst 🌟

Prompt: Wie erstelle ich am besten Diagramme daraus? Ich hab die daten als json und csv datei. Excel? Python?

Reply:

Kurzfassung: Mach die Auswertung und Diagramme in Python, und nutz Excel höchstens zum schnellen Gegencheck oder für Präsentations-Grafiken.

Mit deiner Datenlage (JSON + CSV, viele Dimensionen) ist Python klar überlegen.

Ich sag dir warum, wie, und gebe dir konkrete Plot-Rezepte, die direkt in einer Thesis funktionieren.

1. Excel vs. Python – klare Entscheidungshilfe

Excel – warum eher nicht (für die Hauptarbeit)

- Viele Dimensionen (Alg, Szenario, Stimme, true noise, Optimierung) → unübersichtlich
- Reproduzierbarkeit schlecht
- Fehleranfällig bei Filter/Sortier-Operationen
- Kaum saubere Automatisierung

 Excel ist okay für:

- 1–2 manuelle Abbildungen
- Schnelle Plausibilitätschecks
- Präsentationen

Python – warum es perfekt passt

- JSON & CSV direkt lesbar
- Filter, Gruppierung, Aggregation sauber & reproduzierbar
- Diagramme publikationsreif
- Gleicher Code → gleiche Ergebnisse (wichtig für Thesis!)

Empfohlener Stack:

pandas
matplotlib
seaborn

Empfohlene Minimal-Auswahl für die Arbeit

Wenn du nicht zu viele Abbildungen willst:

1. Balkendiagramm: PESQ (Balanced)
2. Balkendiagramm: STOI (Balanced)
3. Scatter: STOI vs. PESQ
4. Szenarienvergleich (PESQ)
5. Boxplot (PESQ oder STOI)

 5–6 Abbildungen = perfekt

Wenn du willst, kann ich dir:

- ein **komplettes Plot-Skript** für deine Datenstruktur bauen
- helfen, die **Diagramme thesis-tauglich zu beschriften**
- sagen, **welche Plots du lieber weglässt**, um es nicht zu überladen

Sag mir einfach:

👉 „Ich hab die Spalten X, Y, Z – mach mir das Skript“

[ChatGPT-Log]

Datum: 2026-02-16

Uhrzeit: 11:53

Thema/Ziel: Auswertung, Diagramme

Prompt: Das ist mein Evaluierungscode (Anhang Code) Wie stelle ich am besten sowas dar:

#STOI vs. PESQ Trade-off - „Algorithmus A maximiert Qualität, Algorithmus B Verständlichkeit“

#Unterschied von STOI und PESQ Optimierung

Reply:

Für die Darstellung „STOI vs. PESQ Trade-off“ und den Unterschied zwischen **STOI-Optimierung** vs **PESQ-Optimierung** sind 2–3 Plot-Typen am aussagekräftigsten (und sind auch genau die Art, wie man „Qualität vs. Verständlichkeit“ in der Literatur trennt).

1) Der wichtigste Plot: ΔSTOI vs ΔPESQ (pro Datei) + Quadranten

Statt absolute Werte zu plotten, nimm **Verbesserungen gegenüber noisy**:

- $\Delta\text{STOI} = \text{STOI}_{\text{enh}} - \text{STOI}_{\text{noisy}}$
- $\Delta\text{PESQ} = \text{PESQ}_{\text{enh}} - \text{PESQ}_{\text{noisy}}$

Dann als **Scatter** (jeder Punkt = ein Stem).

Damit sieht man sofort:

- Wer primär Verständlichkeit gewinnt (rechts oben / rechts)
- Wer primär Qualität gewinnt (oben / links oben)
- Wer „kaputt verbessert“ (links unten)

Das passt perfekt zu deinem Satz: „Algorithmus A maximiert Qualität, Algorithmus B Verständlichkeit“.

Minimaler Code (neue Funktion)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def plot_tradeoff_delta(df, algs=None, variant="stoiopt", title=None):
    """
    variant: "stoiopt" oder "pesqopt" oder "balopt"
    Plottet  $\Delta\text{STOI}$  vs  $\Delta\text{PESQ}$  pro Algorithmus.
    """
    if algs is not None:
```

```

df = df[df["alg"].isin(algs)].copy()
if df.empty:
    print("No data.")
    return

stoi_col = f"stoi_{variant}"
pesq_col = f"pesq_{variant}"

need = ["stoi_noisy", "pesq_noisy", stoi_col, pesq_col, "alg"]
missing = [c for c in need if c not in df.columns]
if missing:
    print("Missing columns:", missing)
    return

df = df.copy()
df["d_stoi"] = df[stoi_col] - df["stoi_noisy"]
df["d_pesq"] = df[pesq_col] - df["pesq_noisy"]

plt.figure(figsize=(8, 6))
for alg, g in df.groupby("alg"):
    plt.scatter(g["d_stoi"], g["d_pesq"], alpha=0.35, label=f"{alg} (n={len(g)})")

    # Mittelwert als Marker
    plt.scatter([g["d_stoi"].mean()], [g["d_pesq"].mean()], marker="X", s=120)

# Quadrantenlinien
plt.axvline(0, linestyle="--", linewidth=1)
plt.axhline(0, linestyle="--", linewidth=1)

plt.xlabel(f"ΔSTOI = {stoi_col} - stoi_noisy")
plt.ylabel(f"ΔPESQ = {pesq_col} - pesq_noisy")
plt.title(title or f"Trade-off: ΔSTOI vs ΔPESQ ({variant})")
plt.grid(True, linestyle="--", alpha=0.4)
plt.legend()
plt.tight_layout()
plt.show()

```

Wie du es nutzt (z.B. ohne TrueNoise, die drei Algs):

```

df = load_data(lambda name: "ohneTrueNoise" in name)

plot_tradeoff_delta(df, algs=["spectralSubtractor", "wiener", "omlsa"], variant="stoiopt",
                     title="ΔSTOI vs ΔPESQ (STOI-Optimierung) – ohne TrueNoise")

plot_tradeoff_delta(df, algs=["spectralSubtractor", "wiener", "omlsa"], variant="pesqopt",
                     title="ΔSTOI vs ΔPESQ (PESQ-Optimierung) – ohne TrueNoise")

```

Interpretation (für Text in der Arbeit):

- Punkte/Mean **weiter rechts** = intelligibility-getrieben (STOI).
 - Punkte/Mean **weiter oben** = quality-getrieben (PESQ).
 - Ein Algorithmus kann oben sein, aber nicht rechts (Klang wird schöner, aber nicht besser verständlich) – genau diese Trennung wird in der Evaluationsliteratur betont.
-

2) Zweiter Plot: Pareto-Front (pro Algorithmus oder insgesamt)

Wenn du zeigen willst, dass STOI-Opt und PESQ-Opt verschiedene Punkte auf der Pareto-Kurve sind:

- $x = \text{STOI}$, $y = \text{PESQ}$ (oder Δ -Werte)
- Dann **Pareto-Front** markieren (nicht-dominierte Punkte).

Das ist die formalste Trade-off Darstellung.

Skizze (ohne viel Overhead):

```
def pareto_front(points):
    # points: array shape (N,2) mit (stoi, pesq) – wir maximieren beides
    pts = points[np.argsort(-points[:,0])]] # sort by stoi desc
    front = []
    best_pesq = -np.inf
    for s, p in pts:
        if p > best_pesq:
            front.append((s,p))
            best_pesq = p
    return np.array(front)

def plot_pareto(df, alg, title=None):
    d = df[df["alg"] == alg].copy()
    d = d.dropna(subset=["stoi_stoiapt","pesq_stoiapt","stoi_pesqopt","pesq_pesqopt"])

    # Kandidatenpunkte: STOI-opt und PESQ-opt (du kannst balopt auch hinzufügen)
    pts = np.vstack([
        d[["stoi_stoiapt","pesq_stoiapt"]].to_numpy(),
        d[["stoi_pesqopt","pesq_pesqopt"]].to_numpy(),
    ])

    front = pareto_front(pts)

    plt.figure(figsize=(7,6))
    plt.scatter(pts[:,0], pts[:,1], alpha=0.2)
    plt.plot(front[:,0], front[:,1], linewidth=2)
    plt.xlabel("STOI")
    plt.ylabel("PESQ")
    plt.title(title or f"Pareto-Front (STOI vs PESQ) – {alg}")
    plt.grid(True, linestyle="--", alpha=0.4)
    plt.tight_layout()
    plt.show()
```

3) Unterschied STOI-Opt vs PESQ-Opt direkt zeigen (Balken/Box/Violin)

Du willst explizit zeigen: „STOI-Optimierung opfert PESQ“ bzw. umgekehrt.

Mach dafür pro Algorithmus zwei Kennwerte:

- $\Delta\text{STOI}_{\text{stoiapt}}$ vs $\Delta\text{STOI}_{\text{pesqopt}}$
- $\Delta\text{PESQ}_{\text{stoiapt}}$ vs $\Delta\text{PESQ}_{\text{pesqopt}}$

Am besten als **Boxplot** (weil Verteilungen oft schief sind).

Beispiel:

```

def plot_opt_compare(df, algs=None):
    if algs is not None:
        df = df[df["alg"].isin(algs)].copy()

    rows = []
    for _, r in df.iterrows():
        rows.append(("stoiopt", r["alg"], (r["stoi_stoi"]-r["stoi_noisy"])), (r["pesq_stoi"]-r["pesq_noisy"])))
        rows.append(("pesqopt", r["alg"], (r["stoi_pesq"]-r["stoi_noisy"])), (r["pesq_pesq"]-r["pesq_noisy"])))
    d = pd.DataFrame(rows, columns=["opt","alg","d_stoi","d_pesq"])

    plt.figure(figsize=(10,4))
    d.boxplot(column="d_stoi", by=["alg","opt"], rot=45)
    plt.title("ΔSTOI: STOI-Opt vs PESQ-Opt")
    plt.suptitle("")
    plt.axhline(0, linestyle="--", linewidth=1)
    plt.grid(True, linestyle="--", alpha=0.3)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(10,4))
    d.boxplot(column="d_pesq", by=["alg","opt"], rot=45)
    plt.title("ΔPESQ: STOI-Opt vs PESQ-Opt")
    plt.suptitle("")
    plt.axhline(0, linestyle="--", linewidth=1)
    plt.grid(True, linestyle="--", alpha=0.3)
    plt.tight_layout()
    plt.show()

```

4) Was du im Text dazu sauber begründen kannst

- **PESQ ≠ Verständlichkeit:** PESQ ist Qualitäts-/Perzeptions-orientiert und kann bei manchen Enhancement-Artefakten „gut aussehen“, ohne dass Menschen besser verstehen; diese Trennung (Quality vs Intelligibility) wird explizit als zentraler Punkt diskutiert.
-

Kleine Anmerkung zu deinem bestehenden Plot-Code

Dein plot_algorithm_summary() ist für „Durchschnittswerte“ okay, aber für Trade-offs zu grob, weil Mittelwerte **Verteilungen verstecken**. Für Trade-off brauchst du fast immer mindestens einen Plot pro Datei (Scatter/Box).

Wenn du willst, passe ich dir deine plot_algorithm_summary() so an, dass sie optional automatisch:

- Δ-Spalten erzeugt
- Trade-off Scatter + Boxplots + Pareto in einem Rutsch exportiert (PNG + JSON).

Prompt: Integriere 1) in den bestehenden Code

Reply:

Klar — ich integriere dir **Plot 1 (ΔSTOI vs ΔPESQ Scatter + Quadranten + Mittelwert-X)** direkt in deinen bestehenden Evaluierungs-Code-Stil (inkl. folder_filter_func, include_algs, optionaler Filter, title, figsize).

1) Ergänzungen: neue Funktion plot_tradeoff_scatter(...)

Füge **unterhalb** von plot_algorithm_summary(...) (oder irgendwo im selben File) diese Funktion ein:

```

import numpy as np
import matplotlib.pyplot as plt

def plot_tradeoff_scatter(
    folder_filter_func,
    variant="stoiopt",      # "stiopt" | "pesqopt" | "balopt"
    title=None,
    include_algs=None,
    filter_metric=None,
    filter_max=None,
    show_means=True,
    show_origin_lines=True,
    alpha_points=0.25,
    figsize=(8, 6)):
    """
    Trade-off Scatter: ΔSTOI vs ΔPESQ (je Datei ein Punkt), farblich pro Algorithmus.

    ΔSTOI = stoi_<variant> - stoi_noisy
    ΔPESQ = pesq_<variant> - pesq_noisy
    """

    df = load_data(folder_filter_func)
    if df is None:
        return

    # Filter algorithms
    if include_algs is not None:
        df = df[df['alg'].isin(include_algs)]
    if df.empty:
        print("No data left after filtering algorithms.")
        return

    # Optional row filter (z.B. nur schlechte noisy Qualität)
    if filter_metric is not None:
        if filter_metric not in df.columns:
            print(f"Warning: filter_metric '{filter_metric}' not found, skipping filter.")
        else:
            before = len(df)
            if filter_max is not None:
                df = df[df[filter_metric] <= filter_max]
            after = len(df)
            print(f"Filtered by {filter_metric} <= {filter_max}): {after} of {before} rows kept.")
            if after == 0:
                print("No data left after filtering.")
                return

    stoi_col = f"stoi_{variant}"
    pesq_col = f"pesq_{variant}"

    required = ["stoi_noisy", "pesq_noisy", stoi_col, pesq_col, "alg"]
    missing = [c for c in required if c not in df.columns]
    if missing:
        print("Missing required columns:", missing)
        return

    # Compute deltas

```

```

df = df.copy()
df["d_stoi"] = df[stoi_col] - df["stoi_noisy"]
df["d_pesq"] = df[pesq_col] - df["pesq_noisy"]

# Drop NaNs (falls irgendwo PESQ/STOI fehlt)
df = df.dropna(subset=["d_stoi", "d_pesq", "alg"])
if df.empty:
    print("No valid rows after dropping NaNs.")
    return

plt.figure(figsize=figsize)

# Scatter per algorithm
for alg, g in df.groupby("alg"):
    plt.scatter(g["d_stoi"], g["d_pesq"], alpha=alpha_points, label=f"{alg} (n={len(g)})")

# Mean marker
if show_means and len(g) > 0:
    plt.scatter([g["d_stoi"].mean()], [g["d_pesq"].mean()], marker="X", s=140)

if show_origin_lines:
    plt.axvline(0, linestyle="--", linewidth=1)
    plt.axhline(0, linestyle="--", linewidth=1)

plt.xlabel(f" $\Delta\text{STOI} = \{\text{stoi\_col}\} - \text{stoi\_noisy}$ ")
plt.ylabel(f" $\Delta\text{PESQ} = \{\text{pesq\_col}\} - \text{pesq\_noisy}$ ")
plt.title(title or f"Trade-off:  $\Delta\text{STOI}$  vs  $\Delta\text{PESQ}$  ({variant})")
plt.grid(True, linestyle="--", alpha=0.4)
plt.legend()
plt.tight_layout()
plt.show()

```

3) Kleine, aber wichtige Empfehlung (damit's „wissenschaftlicher“ wirkt)

Wenn du *noch klarer* zeigen willst „A = Qualität, B = Verständlichkeit“, dann nutze **denselben Plot zweimal** (einmal `variant="stoipt"`, einmal `variant="pesqopt"`) und beschreibe:

- **STOI-Optimierung** verschiebt die Wolke tendenziell nach rechts ($\Delta\text{STOI} \uparrow$)
- **PESQ-Optimierung** verschiebt sie tendenziell nach oben ($\Delta\text{PESQ} \uparrow$)

Wenn du willst, kann ich dir als nächsten Schritt noch 2 Zeilen ergänzen, die automatisch pro Algorithmus (**mean ΔSTOI , mean ΔPESQ**) in der Konsole ausgeben (als Mini-Tabelle) – das ist oft super für den Ergebnisteil.

Prompt: Was bedeuten die Kreuze? Dafür wird auch eine Legende benötigt

Reply:

Sehr gute Frage 

Die **Kreuze (Marker „X“)** sind aktuell:

Der Mittelwert (Mean) von ΔSTOI und ΔPESQ pro Algorithmus

Also:

- Die vielen transparenten Punkte = einzelne Dateien
- Das große „X“ = durchschnittliche Verbesserung dieses Algorithmus

Ohne Erklärung wirkt das aber wie „zufällige Marker“. Deshalb solltest du sie **explizit in die Legende aufnehmen**.

Saubere Lösung: Mittelwert mit eigener Legende

Ersetze in der Funktion plot_tradeoff_scatter() den Scatter-Teil durch diese Version:

```
plt.figure(figsize=figsize)

mean_handles = []

for alg, g in df.groupby("alg"):
    # Punkte (Einzelfiles)
    sc = plt.scatter(
        g["d_stoi"],
        g["d_pesq"],
        alpha=alpha_points,
        label=f"{alg} (files)"
    )

    # Mittelwert
    if show_means and len(g) > 0:
        mean_point = plt.scatter(
            [g["d_stoi"].mean()],
            [g["d_pesq"].mean()],
            marker="X",
            s=160,
            edgecolors="black",
            linewidths=1.2,
            label=f"{alg} mean"
        )
```

Damit bekommt jedes Kreuz einen eigenen Legenden-Eintrag wie:

```
spectralSubtractor (files)
spectralSubtractor mean
wiener (files)
wiener mean
...
```

Noch besser (wissenschaftlich sauberer)

Wenn du es übersichtlicher willst, kannst du:

- Punkte = farbig
- Mittelwert-X = gleiche Farbe, aber größer + schwarzer Rand

Dann erkennt man sofort:

→ „Das X gehört zu dieser Punktfolge“

Optional: Noch klarer machen

Du kannst die Beschriftung oben auch ergänzen:

```
plt.title(title or f"Trade-off: ΔSTOI vs ΔPESQ ({variant})\nX = Algorithmus-Mittelwert")
```

Interpretation für deine Arbeit

Dann kannst du im Text schreiben:

Die transparenten Punkte repräsentieren die Verbesserungen einzelner Sprachdateien.

Das Kreuz markiert den mittleren Verbesserungsvektor des jeweiligen Algorithmus im ΔSTOI–ΔPESQ-Raum.

Das ist exakt die Art von Darstellung, die bei Trade-off-Analysen üblich ist.

Wenn du möchtest, kann ich dir noch eine Version machen, bei der:

- nur **ein einziges großes Kreuz pro Algorithmus** gezeigt wird
- und die Einzelfiles nur schwach im Hintergrund liegen
→ wirkt oft noch professioneller für Publikation.

Prompt: Jetzt möchte ich noch zeigen, wie der Unterschied von pesqopt, stoiopt und balopt ist

Kann ich das in einem diagramm zeigen?

Reply:

Ja — das geht sehr gut in **einem** Diagramm (bzw. in *einer* Figure), ohne dass es unlesbar wird. Die zwei besten Optionen:

Option A (meine Empfehlung): "3 Mittelwert-Punkte pro Algorithmus" im ΔSTOI–ΔPESQ Raum

Du plottest weiterhin **ΔSTOI vs ΔPESQ**, aber **nicht alle Files**, sondern nur die **3 Mittelwerte** pro Algorithmus:

- stoiopt = Punkt 1
- pesqopt = Punkt 2
- balopt = Punkt 3
und verbindest sie mit Linien → dann sieht man sofort den „Shift“ von STOI→BAL→PESQ.

Option B: 2 Panels in einer Figure (übersichtlicher für viele Punkte)

Wenn du alle Files behalten willst, ist „alles in ein Plot“ oft zu voll. Dann mach 1 Figure mit 3 Subplots (stoiopt/balopt/pesqopt) — wirkt immer noch wie ein Diagramm/Abbildung.

Wenn du willst, baue ich dir dafür eine `plot_tradeoff_variants_subplots(...)` Funktion.

Wenn du sagst, ob du **Mittelwerte** reichen (Option A) oder ob du unbedingt **alle Punkte** sehen willst (Option B), passe ich es dir direkt passend an deinen Code an.

Prompt: Was sagt das Diagramm aus? (Anhang Bild Diagramm)

Reply:

Das Diagramm ist eine **Trade-off-Karte der Mittelwerte** – pro Algorithmus siehst du **drei “Betriebspunkte”**:

- **x-Achse: ΔSTOI** (rechts = Verständlichkeit besser als noisy, links = schlechter)
- **y-Achse: ΔPESQ** (oben = Qualität besser als noisy, unten = schlechter)
- Die gestrichelten Linien bei 0 sind die „kein Unterschied“-Grenzen.
- **Jede Linie** verbindet beim selben Algorithmus: pesqopt → balopt → stoiopt.

Was es in deinem Plot konkret aussagt

1) Klarer Zielkonflikt: Qualität vs. Verständlichkeit

Für alle drei Algorithmen gilt im Mittel:

- **pesqopt** liegt **weit oben**, aber **links von 0**
→ Qualität stark besser, aber Verständlichkeit im Mittel schlechter.
- **stoiopt** liegt **rechts von 0**, aber deutlich **tiefer**
→ Verständlichkeit etwas besser, aber viel weniger Qualitätsgewinn.
- **balopt** liegt **dazwischen** – aber immer noch **links von 0**
→ Kompromiss: fast so viel PESQ wie pesqopt, aber STOI-Verlust etwas reduziert – jedoch im Mittel noch negativ.

Kurz: Du „kaufst“ PESQ mit STOI ein.

2) „balopt“ ist bei dir eher PESQ-lastig (nicht wirklich neutral)

Dass **balopt** bei allen Algorithmen **links von $\Delta\text{STOI} = 0$** liegt, heißt:

Dein Balancing-Ziel wählt im Mittel Settings, die **STOI verschlechtern**, obwohl sie PESQ stark verbessern.

Wenn du „balopt“ wirklich als *Kompromiss ohne STOI-Verlust* willst, müsstest entweder:

- die **Gewichtung/Score-Funktion** mehr STOI priorisieren, oder
 - du setzt eine **Nebenbedingung** wie „ $\Delta\text{STOI} \geq 0$ “ und optimierst dann PESQ.
-

3) Unterschiede zwischen den Algorithmen

- **Wiener**: hat bei **stoiopt** schon den höchsten ΔPESQ (dein wiener-stoiopt Punkt ist höher als die anderen) → Wiener schafft Qualität, ohne STOI-Opt komplett „qualitätblind“ zu sein.
- **OMLSA und SpectralSubtractor**: stoßen bei **stoiopt** in PESQ eher niedrig aus → die STOI-Optimierung bringt dort wenig Qualitätsgewinn.

Bei **pesqopt** scheint Wiener am stärksten nach links zu gehen (größter STOI-Verlust), aber auch sehr hoch (viel PESQ).

4) Was du daraus als 1-Satz-Interpretation schreiben kannst

„Die Mittelwert-Trajektorien zeigen einen ausgeprägten Trade-off: PESQ-Optimierung erhöht die Qualität stark, führt jedoch im Mittel zu negativer ΔSTOI ; die balancierte Optimierung reduziert den STOI-Verlust gegenüber PESQ-Optimierung bei nur geringem PESQ-Abfall, bleibt aber insgesamt STOI-negativ.“

Wenn du willst, kann ich dir noch eine kleine Kennzahl dazu geben, die du direkt aus diesen drei Punkten ableitest (z.B. „**PESQ-Gewinn pro STOI-Verlust**“ von pesqopt→balopt und balopt→stoiopt). Damit wird die Aussage noch quantitativer.

[ChatGPT-Log]

Datum: 2026-02-18

Uhrzeit: 15:05

Thema/Ziel: Prüfung Text

Prompt: Prüfe meine Arbeit inhaltlich, vergleiche mit den quellen (Anhang PDF)

Reply:

Hier kommt eine **inhaltliche Fachprüfung mit klarer Einordnung: korrekt / präzisieren / kritisch / sehr gut.**

1 Grundmodell & STFT-Rahmen

Additives Modell

Du verwendest:

$$y(n) = x(n) + d(n)$$

Das entspricht exakt der Standardannahme in

Speech Enhancement: Theory and Practice (Kap. 1) sowie bei **Yariv Ephraim & David Malah** (1984).

- ✓ Fachlich korrekt.
- ✓ STFT-basierte frameweise Verarbeitung ist Standard.

Empfehlung:

Explizit erwähnen, dass innerhalb eines Frames Quasi-Stationarität angenommen wird (20–40 ms), wie bei Ephraim & Malah 1984 beschrieben.

2 Spektrale Subtraktion

Literaturvergleich

Element	Deine Arbeit	Literatur
Subtraktion im Leistungsspektrum	✓	Steven F. Boll 1979
Oversubtraction α	✓	M. Berouti 1979
Spectral Floor β	✓	Berouti 1979
Musical Noise erwähnt	✓	Boll 1979

Fachliche Bewertung

- ✓ Mathematisch korrekt
- ✓ Parameterwahl entspricht Literatur
- ✓ Zusammenhang $\alpha \leftrightarrow$ Artefakte korrekt

Kleiner Präzisierungspunkt

Bei Boll 1979 wird ursprünglich im **Betragsspektrum** gearbeitet.

Berouti geht explizit ins **Leistungsspektrum** (Power Spectral Subtraction).

Dein Code:

```
power_clean = power_noisy - alpha * power_noise  
power_clean = np.maximum(power_clean, beta * power_noise)
```

Das ist korrekt nach Berouti.

☞ In der Arbeit klar formulieren:

„Implementiert wurde die Power Spectral Subtraction nach Berouti (1979)“

3 Wiener-Filter

Literaturbasis

Wiener Gain:

$$G = \frac{\xi}{1 + \xi}$$

Das ist exakt der klassische STFT-Wienerfilter wie in:

- **Speech Enhancement: Theory and Practice**
- **Yariv Ephraim 1984**

Decision-Directed Ansatz

$$\xi_k = \alpha G_{k-1}^2 \gamma_{k-1} + (1 - \alpha) \max(\gamma_k - 1, 0)$$

✓ Genau nach Ephraim & Malah 1984.

Dein Code:

```
recursive = (prev_gain ** 2) * prev_gamma  
ksi = alpha * recursive + (1.0 - alpha) * direct
```

✓ Formal korrekt.

Bewertung

- ✓ Mathematisch exakt
 - ✓ Parameterwahl (α 0.85–0.99) typisch
 - ✓ Gain Floor → praktisch sinnvoll
-

4 MMSE-STSA (Ephraim & Malah 1984)

Literatur

Yariv Ephraim &

David Malah,

„Speech Enhancement Using a Minimum Mean-Square Error Short-Time Spectral Amplitude Estimator“, 1984

Deine Implementierung

Du nutzt exakt:

$$G(v) = \frac{\sqrt{\pi}}{2} \frac{\sqrt{v}}{\gamma} e^{-v/2} [(1 + v) I_0(v/2) + v I_1(v/2)]$$

✓ Das ist exakt die Originalformel.

✓ Du begrenzt v numerisch → korrekt.

✓ Decision-Directed korrekt integriert.

Bewertung

- Wissenschaftlich sehr sauber
 - Keine inhaltlichen Fehler
 - Vollständig literaturkonform
-

5 Advanced MMSE / „OMLSA“

Hier wird es fachlich spannend.

Literaturreferenz

Israel Cohen,

„Optimal Speech Enhancement Under Signal Presence Uncertainty Using Log-Spectral Amplitude Estimator“, 2002.

OM-LSA beinhaltet:

1. Log-MMSE Gain
 2. Speech Presence Probability (SPP)
 3. Soft decision weighting
 4. Geometrisches Mittel zweier Gains
-

Vergleich mit deinem Code

Du implementierst:

- Log-MMSE (über Exponentialintegral expn)
- Decision Directed
- SPP über Parameter q
- Noise smoothing noise_mu
- Gain floor

Das entspricht **Cohen 2002** konzeptionell.

Wichtig

☞ Du implementierst **nicht exakt das vollständige OM-LSA aus Cohen 2002**, sondern eine vereinfachte Version mit SPP-Modifikation.

Das ist wissenschaftlich völlig legitim —
aber du solltest es korrekt benennen.

Empfehlung für die Arbeit

Statt „OM-LSA“ besser formulieren:

„LSA-MMSE mit Speech Presence Probability nach Cohen (2002)“

Wenn du es OM-LSA nennst, dann bitte klarstellen:

„Eine vereinfachte Implementierung des OM-LSA-Prinzips“

6 Noise Estimation

Du verwendest:

- Percentile
- Min-Tracking
- True Noise (Oracle)

Das ist didaktisch sehr sauber.

Min-Tracking entspricht klassischen Verfahren aus:

- Loizou 2013
- Cohen 2002 (Noise smoothing)

- ✓ Fachlich korrekt
 - ✓ Realistisch
 - ✓ Literaturkonform
-

7 Objektive Metriken

STOI

Cees H. Taal, 2011

- ✓ Du nutzt extended=False → korrekt für klassische STOI
 - ✓ Framebasierte Intelligibilitätsmessung → passend
-

PESQ

ITU-T P.862

- ✓ 16 kHz wideband → korrekt
 - ✓ Resampling korrekt implementiert
-

Literaturabgleich

Wichtig:

Yi Hu & Philipos C. Loizou zeigen:

Qualitätsverbesserung ≠ Intelligibilitätsverbesserung

Deine Arbeit zeigt genau diesen Trade-off.

↳ Das passt perfekt zur Literatur.

8 Deine Beobachtung: „Alle Algorithmen sind ähnlich gut“

Das ist absolut realistisch.

Hu & Loizou 2007 zeigen:

- Wiener, Log-MMSE, Subtraktion → ähnliche Intelligibilitätswerte
- Unterschiede eher in Qualität als in STOI

Warum?

Weil:

- Alle Gains basieren auf ähnlicher SNR-Schätzung
- Noise estimation dominiert Performance
- Decision Directed glättet Unterschiede

Deine Ergebnisse sind daher **sehr plausibel**.

Fachliche Gesamtbewertung

Bereich	Bewertung
Theoretische Herleitung	
Literaturtreue	
Implementierung	
Parameterauswahl	
Interpretation	
OMLSA-Nennung	Präzisieren

Was solltest du noch verbessern?

- 1. OMLSA sauber benennen**
 - 2. Musical Noise theoretisch einordnen**
 - 3. Decision Directed explizit als Ephraim & Malah zitieren**
 - 4. Klarer formulieren, dass Unterschiede gering sind laut Literatur**
-

Gesamturteil

Deine Arbeit ist:

- mathematisch korrekt
 - literaturkonform
 - methodisch sauber
 - experimentell nachvollziehbar
 - konsistent mit Hu & Loizou
-

Wenn du möchtest, kann ich dir jetzt noch:

- eine „Prüfer-Simulation“ schreiben (kritische Fragen)
- prüfen, ob deine Interpretation wissenschaftlich scharf genug ist
- dein Fazit wissenschaftlich „hochziehen“
- konkrete Zitatstellen aus Loizou markieren, die du verwenden kannst

Was möchtest du als Nächstes?