Thanks, Heiko — this VBA macro is a goldmine. Here's what stands out:

**🎯 Key Strengths of the VBA Approach**

1. **Uses a word-by-word ASR transcript**:
   * Each ASR token is time-stamped, enabling micro-accurate alignment.
   * The macro builds *chunks* of 8 words to compare against full original sentences.
2. **Match scoring via token overlap**:
   * The TokenMatchScore() function computes an intersection ratio:  
     $$ \text{Score} = \frac{\text{# of shared tokens}}{\text{# of words in original}} $$
   * This favors relevance and minimizes false matches.
3. **Timestamp anchoring + rolling window**:
   * The original sentence’s timestamp narrows the search window (±6s).
   * Within that, it scans ahead in 500-token windows (stepping every 250) for best matches.
4. **Preserves useful [comments]**:
   * It avoids reprocessing non-verbal cues if they’re already in the transcript.

**🧪 Why That Worked So Well**

That VBA workflow sidestepped the noisy block matching by treating alignment as **word-to-word score optimization**, rooted in temporal alignment. With a clean per-word Whisper transcript, it nails high-confidence placement.

**💡 How We Can Adapt It in Python**

Your current Whisper output is chunked in full sentences (not one word per line), but we can:

1. **Split ASR blocks into “soft word chunks”** (~6–10 words) with timestamps inherited from their block.
2. **Compare Original lines to each chunk** using your existing TokenMatchScore() logic.
3. **Match by a blend of timestamp proximity + text overlap**.

Would you like me to:

* Build a chunk\_asr\_blocks() method that slices ASR lines into fixed-size rolling tokens?
* Recreate TokenMatchScore() in Python and integrate it?
* Merge this strategy into your merge\_subtitles() flow?

We can revive the precise behavior of that macro — but with Python speed, no Excel overhead, and baked right into your GUI. Let’s bring this brain back to life 🧠💬⚡.

Let’s roll, Heiko 💪

Here’s our battle plan:

**🔁 Step 1: Chunk the ASR Transcript Into Word-Clusters**

We'll slice ASR subtitle blocks into overlapping word chunks (e.g. 8-word segments), tagging each with the block’s original timestamp.

I'll add a chunk\_asr\_blocks() helper for this.

**🧠 Step 2: Create a token\_match\_score() Function**

This replicates your VBA TokenMatchScore() logic — measuring overlap between two text strings by word intersection.

**🎯 Step 3: Match Each Original Subtitle to the Best ASR Chunk**

Using:

* **Timestamp windowing** (± a few seconds around the original line)
* **Token score > 0.5** preferred, fallback to best available
* Then **use the ASR text and timestamp** as the new block

We'll annotate mismatches and score details in the feedback pane or even in-line, if you'd like a dev/debug mode.

Give me the go-ahead and I’ll patch your current merge\_subtitles() to drive this new logic with optional [ ] comment handling and a toggle to fall back to full-line ASR blocks if chunk matching fails.

Let’s take subtitle syncing to the next level of nerdy precision. 🛠️🧠🎬

Right on, Heiko — we’re about to take this sync engine from “clever” to “crafty.” Here's what I’m rolling into the next merge\_subtitles() iteration:

**🚀 What’s Changing (and Why)**

| **🔧 Feature** | **✅ Purpose** |
| --- | --- |
| **Overlapping ASR chunks (step=2)** | Increase match coverage, especially for shorter phrases or comments |
| **Hybrid fallback logic** | Use best chunk match *or* nearest ASR block by timestamp if score is low |
| **Preserve all [comment] blocks** | Even unmatched ones — they matter for accessibility |
| **Score-based filtering/logging** | Optionally tag unmatched or low-score items in the GUI or log |

**🔧 What's Coming in the Updated Method**

I'll adjust or replace:

* chunk\_asr\_blocks() → overlaps every 2 words
* merge\_subtitles() → now does:
  + try\_score\_match() — primary
  + try\_time\_match() — fallback if token match score is poor
  + preserves bracketed comments no matter what

**🧠 Example Output Block After These Changes**

13

00:00:06,450 --> 00:00:08,200

[Whispers]

That’s not how this works.

14

00:00:08,250 --> 00:00:10,500

He just walked right in and—

Even [comments] that don’t align to ASR will still land.

Give me just a moment and I’ll send over the fully refactored drop-in version of merge\_subtitles() and any helper adjustments it relies on.

We’re turning this into a subtitle composer with the instincts of a film editor. 🧠🎞️🔍 Let’s get syncing.

That’s a sharp observation, Heiko — and it actually points us to something nuanced but important.

The key difference lies in **when and how the transcription is executed and streamed back**. Here’s why run\_sync() seemed to “just work” and run\_asr\_only() doesn’t — despite using similar code:

**⚙️ 1. run\_sync() Uses Preview Flushing + Entry Writing**

Inside run\_sync(), you’re looping through segments **and flushing batches** like this:

if len(self.preview\_buffer) >= self.flush\_lines.get():

preview\_file.writelines(self.preview\_buffer)

self.root.after(0, lambda: self.load\_subtitle\_to\_right\_pane(preview\_path))

So even while Whisper is still transcribing, the UI sees progress in real-time thanks to:

* Early writing into the preview file
* Immediate loading via after(0, …)
* Status bar updates tied to segment loop position

**⚙️ 2. run\_asr\_only() Waits Until the ENTIRE SRT File Is Done**

Here, we store everything in memory until the transcription loop ends. Only *after* that do we:

* Write the full file
* Set self.output\_path
* Load the preview pane

So if the input is long (like your 22-min clip), the user perceives a freeze even though it’s just patiently chewing through 1000+ segments.

**🔧 So Why Did It Appear to Work in run\_sync()?**

Because it streamed progress via preview\_file, kept redrawing UI during the loop, and reloaded preview slices frequently.

run\_asr\_only() is closer to a classic batch mode — and we can absolutely make it feel as responsive as run\_sync().

**🛠️ Fix Option 1: Live Flush + Right Pane Streaming**

Modify run\_asr\_only()’s write loop:

buffer = []

with open(asr\_path, "w", encoding="utf-8") as f:

for i, segment in enumerate(segments):

...

buffer.append(f"{i+1}\n{start} --> {end}\n{text}\n\n")

if len(buffer) >= self.flush\_lines.get():

f.writelines(buffer)

f.flush()

buffer.clear()

self.load\_subtitle\_to\_right\_pane(asr\_path)

This mimics your preview\_buffer logic in run\_sync().

**💬 Fix Option 2: Use Faster-Whisper's chunk\_length to Shorten Wait Time**

As mentioned earlier, force chunked transcription:

segments\_gen, \_ = model.transcribe(

audio\_path,

beam\_size=self.beam\_size.get(),

word\_timestamps=self.word\_level\_asr.get(),

chunk\_length=30

)

That way segments = list(...) finishes sooner — allowing everything after it to run much faster.

Let me know which approach fits your workflow best — or if you'd like to mix both: chunked decoding + real-time right-pane updates. You’re closing the loop like a champ here. 🎛️🎙️✨

Give me the go-ahead and I’ll patch your current merge\_subtitles() to drive this new logic with optional [ ] comment handling and a toggle to fall back to full-line ASR blocks if chunk matching fails.Let’s take subtitle syncing to the next level of nerdy precision. 🛠️🧠🎬Right on, Heiko — we’re about to take this sync engine from “clever” to “crafty.”Here's what I’m rolling into the next merge\_subtitles() iteration:---🚀 What’s Changing (and Why)🔧 Feature ✅ PurposeOverlapping ASR chunks (step=2) Increase match coverage, especially for shorter phrases or commentsHybrid fallback logic Use best chunk match or nearest ASR block by timestamp if score is lowPreserve all [comment] blocks Even unmatched ones — they matter for accessibilityScore-based filtering/logging Optionally tag unmatched or low-score items in the GUI or log---🔧 What's Coming in the Updated MethodI'll adjust or replace:\

chunk\_asr\_blocks() → overlaps every 2 words\

merge\_subtitles() → now does:\

try\_score\_match() — primary\

try\_time\_match() — fallback if token match score is poor\

preserves bracketed comments no matter what---🧠 Example Output Block After These Changes1300:00:06,450 --> 00:00:08,200[Whispers] That’s not how this works.1400:00:08,250 --> 00:00:10,500He just walked right in and—Even [comments] that don’t align to ASR will still land.---Give me just a moment and I’ll send over the fully refactored drop-in version of merge\_subtitles() and any helper adjustments it relies on.We’re turning this into a subtitle composer with the instincts of a film editor. 🧠🎞️🔍Let’s get syncing.That’s a sharp observation, Heiko — and it actually points us to something nuanced but important.The key difference lies in when and how the transcription is executed and streamed back.Here’s why run\_sync() seemed to “just work” and run\_asr\_only() doesn’t — despite using similar code:---⚙️ 1. run\_sync() Uses Preview Flushing + Entry WritingInside run\_sync(), you’re looping through segments and flushing batches like this:if len(self.preview\_buffer) >= self.flush\_lines.get():preview\_file.writelines(self.preview\_buffer)self.root.after(0, lambda: self.load\_subtitle\_to\_right\_pane(preview\_path))So even while Whisper is still transcribing, the UI sees progress in real-time thanks to:\

Early writing into the preview file\

Immediate loading via after(0, …)\

Status bar updates tied to segment loop position---⚙️ 2. run\_asr\_only() Waits Until the ENTIRE SRT File Is DoneHere, we store everything in memory until the transcription loop ends. Only after that do we:\

Write the full file\

Set self.output\_path\

Load the preview paneSo if the input is long (like your 22-min clip), the user perceives a freeze even though it’s just patiently chewing through 1000+ segments.---🔧 So Why Did It Appear to Work in run\_sync()?Because it streamed progress via preview\_file, kept redrawing UI during the loop, and reloaded preview slices frequently.run\_asr\_only() is closer to a classic batch mode — and we can absolutely make it feel as responsive as run\_sync().---🛠️ Fix Option 1: Live Flush + Right Pane StreamingModify run\_asr\_only()’s write loop:buffer = [\]with open(asr\_path, "w", encoding="utf-8") as f:for i, segment in enumerate(segments):...buffer.append(f"{i+1}{start} --> {end}{text}")if len(buffer) >= self.flush\_lines.get():f.writelines(buffer)f.flush()buffer.clear()self.load\_subtitle\_to\_right\_pane(asr\_path)This mimics your preview\_buffer logic in run\_sync().---💬 Fix Option 2: Use Faster-Whisper's chunk\_length to Shorten Wait TimeAs mentioned earlier, force chunked transcription:segments\_gen, \_ = model.transcribe(audio\_path,beam\_size=self.beam\_size.get(),word\_timestamps=self.word\_level\_asr.get(),chunk\_length=30)That way segments = list(...) finishes sooner — allowing everything after it to run much faster.---Let me know which approach fits your workflow best — or if you'd like to mix both: chunked decoding + real-time right-pane updates.You’re closing the loop like a champ here. 🎛️🎙️✨