**1) “Where does the mic indicator hide—nothing in the systray?”**

The overlay I built is *just* the full-screen visualization. It doesn’t create or manage a tray icon. Your **toolbar** (your main tiny always-on-top window) should be the thing that hides, and your **app** needs a QSystemTrayIcon that stays visible while recording. If you don’t already create a tray icon, you won’t see anything in the systray.

**2) “How do I get the mic indicator back once I pressed on the mic?”**

You show the overlay again when you start recording again. Typical triggers:

* Click a **“Start recording”** item in the tray menu, or
* Press a **global shortcut** you bind in your app (or at least a normal QShortcut while the toolbar is focused).  
  In code terms: call overlay.show\_overlay() when you start recording, and overlay.hide\_overlay() when you stop.

**3) “How do I close the app?”**

Add a **Quit** action to the tray menu that calls QApplication.quit(). Also set QApplication.setQuitOnLastWindowClosed(False) so the app keeps running when the toolbar is hidden.

**Minimal wiring (drop-in)**

This glues your existing toolbar + recorder with the canvas component (RecordingOverlay).

from PyQt5 import QtCore, QtGui, QtWidgets

from RecordingOverlay import RecordingOverlay # from the canvas file

class Controller(QtCore.QObject):

def \_\_init\_\_(self, toolbar: QtWidgets.QWidget, parent=None):

super().\_\_init\_\_(parent)

self.app = QtWidgets.QApplication.instance()

self.toolbar = toolbar

self.overlay = RecordingOverlay(show\_waveform=True)

self.overlay.stopRequested.connect(self.\_stop\_recording\_from\_overlay)

# ——— System Tray ———

self.app.setQuitOnLastWindowClosed(False)

self.tray = QtWidgets.QSystemTrayIcon(self)

self.tray.setIcon(toolbar.windowIcon() or self.app.windowIcon() or QtGui.QIcon()) # set a proper icon!

self.tray.setToolTip("Dictation is ready")

menu = QtWidgets.QMenu()

act\_show\_toolbar = menu.addAction("Show Toolbar")

act\_start = menu.addAction("Start Recording")

act\_stop = menu.addAction("Stop Recording")

menu.addSeparator()

act\_quit = menu.addAction("Quit")

act\_show\_toolbar.triggered.connect(self.\_show\_toolbar)

act\_start.triggered.connect(self.start\_recording)

act\_stop.triggered.connect(self.stop\_recording)

act\_quit.triggered.connect(QtWidgets.QApplication.quit)

self.tray.setContextMenu(menu)

self.tray.activated.connect(self.\_tray\_activated) # left click toggles toolbar, optional

self.tray.show()

# ---- Public API you can call from your mic button on the toolbar ----

def start\_recording(self):

# 1) hide toolbar to “systray”

self.toolbar.hide()

# 2) start your audio pipeline here

# self.recorder.start()

# 3) show overlay

self.overlay.set\_hint\_text("Listening… Tap to stop")

self.overlay.show\_overlay()

def stop\_recording(self):

# 1) stop your audio pipeline

# self.recorder.stop()

# 2) hide overlay (if still visible)

self.overlay.hide\_overlay()

# 3) restore toolbar

self.\_show\_toolbar()

# ---- Slots & helpers ----

def \_stop\_recording\_from\_overlay(self):

# User clicked the overlay or pressed Esc/Space/Enter

self.stop\_recording()

def \_show\_toolbar(self):

self.toolbar.show()

self.toolbar.raise\_()

self.toolbar.activateWindow()

def \_tray\_activated(self, reason):

# Optional: left-click tray toggles toolbar

if reason == QtWidgets.QSystemTrayIcon.Trigger:

if self.toolbar.isVisible():

self.toolbar.hide()

else:

self.\_show\_toolbar()

# Optionally, call this from your audio meter at ~30–60 Hz (0..1)

def update\_level(self, level: float):

self.overlay.update\_level(level)

**How you use it**

* Create this controller once, pass your toolbar widget into it.
* Wire your **toolbar mic button** to controller.start\_recording().
* When your recording logic decides to stop, call controller.stop\_recording() (or let the overlay stop via user click/keys).
* Feed VU levels to controller.update\_level(level) if you want the waveform to animate.

**Checklist**

* Ensure you actually **create the tray icon** (QSystemTrayIcon) and call tray.show().
* Set a **real icon** (PNG/ICO) so the tray is visible on Windows.
* Add **Quit** in the tray menu to exit cleanly.
* Consider a **global shortcut** for “Start recording” if the toolbar is hidden (Windows: a small native hook, or a third-party lib; within pure Qt, a QShortcut works only when a window is focused).

If you want, I can fold this Controller into the canvas as a small companion file and adapt names to your real classes; just tell me your toolbar class name and how you start/stop the mic pipeline.

**You said:**

How do I Feed VU levels to controller.update\_level(level)

**ChatGPT said:**

Thought for 4 seconds

Great—controller.update\_level(level) just wants a float in **[0..1]** (RMS/peak). Here are two clean ways to produce that and wire it in.

**Option A — Using QAudioInput (all-Qt)**

from PyQt5 import QtCore, QtMultimedia

import numpy as np

class AudioLevelEmitter(QtCore.QObject):

levelChanged = QtCore.pyqtSignal(float)

def \_\_init\_\_(self, parent=None, device=None, sample\_rate=16000, channels=1):

super().\_\_init\_\_(parent)

fmt = QtMultimedia.QAudioFormat()

fmt.setSampleRate(sample\_rate)

fmt.setChannelCount(channels)

fmt.setSampleSize(16)

fmt.setCodec("audio/pcm")

fmt.setByteOrder(QtMultimedia.QAudioFormat.LittleEndian)

fmt.setSampleType(QtMultimedia.QAudioFormat.SignedInt)

if device is None:

device = QtMultimedia.QAudioDeviceInfo.defaultInputDevice()

if not device.isFormatSupported(fmt):

fmt = device.nearestFormat(fmt)

self.audio = QtMultimedia.QAudioInput(device, fmt, self)

self.io = self.audio.start()

self.timer = QtCore.QTimer(self)

self.timer.setInterval(33) # ~30 Hz

self.timer.timeout.connect(self.\_poll)

self.timer.start()

self.\_ema = 0.0 # smoothing

def \_poll(self):

avail = self.audio.bytesReady()

if avail <= 0:

self.levelChanged.emit(max(0.0, min(1.0, self.\_ema)))

return

data = self.io.read(avail)

if not data:

return

# Convert PCM16LE -> numpy

arr = np.frombuffer(data, dtype=np.int16)

if arr.size == 0:

return

# mono mix if needed

# (Qt already gives interleaved channels; for >1 channels, average them)

rms = np.sqrt(np.mean((arr.astype(np.float32) / 32768.0) \*\* 2))

# light compression to feel more responsive on speech

level = min(1.0, rms \*\* 0.5)

# exponential smoothing

alpha = 0.25

self.\_ema = (1 - alpha) \* self.\_ema + alpha \* level

self.levelChanged.emit(self.\_ema)

**Wire it to the controller:**

self.leveler = AudioLevelEmitter()

self.leveler.levelChanged.connect(self.update\_level) # calls controller.update\_level

**Option B — From your existing recorder thread/callback**

If you already capture audio elsewhere (PyAudio, sounddevice, custom ASR client), compute RMS there and emit a Qt signal:

import numpy as np

from PyQt5 import QtCore

class MeterBridge(QtCore.QObject):

levelChanged = QtCore.pyqtSignal(float)

def push\_pcm16le(self, data\_bytes):

arr = np.frombuffer(data\_bytes, dtype=np.int16)

if arr.size == 0:

return

rms = np.sqrt(np.mean((arr.astype(np.float32)/32768.0) \*\* 2))

level = min(1.0, rms \*\* 0.5)

self.levelChanged.emit(level)

In your audio callback (any thread):

# audio\_callback(data\_bytes):

QtCore.QMetaObject.invokeMethod(

meter\_bridge,

"push\_pcm16le",

QtCore.Qt.QueuedConnection,

QtCore.Q\_ARG(bytes, data\_bytes)

)

Then:

meter\_bridge.levelChanged.connect(controller.update\_level)

**Notes & tweaks**

* Use **RMS** for steadiness; **peak** (np.max(np.abs(arr))/32768.0) is twitchier.
* The \*\*0.5 “soft-knee” brings quiet speech into a nicer visual range—drop it if you prefer linear.
* Call rate: **20–60 Hz** feels smooth with the overlay’s waveform.
* If your audio is **float32** in [-1, 1], skip the /32768.0 and dtype cast.

If you tell me which audio stack you use (Qt, PyAudio, sounddevice, Riva/NVIDIA, etc.), I’ll tailor the snippet exactly to it.

Top of Form

Bottom of Form