ChatGPT

Echo V4.0 AGI System Codebase

The Echo V4.0 system integrates multiple advanced modules. Here is an overview of each component, with references to the provided design and code documents. Each module is assumed written in Python 3 and orchestrated in a real-time loop (no external servers).

VoiceCrystal (Voice Cloning & Prosody Engine)

Function: Wraps a TTS system to mimic the child's voice across styles and modes (outer/inner/coach) with prosody transfer. It maintains a small "voice profile" of reference samples (facets) so that the synthesized speech matches the child's timbre.

- **VoiceProfile & Facets:** A local library of labeled voice samples (e.g. `calm_1.wav`, `excited_1.wav`, etc.) is managed by a `VoiceProfile` class. When speaking, `VoiceCrystal` picks a reference sample for the requested style and updates the TTS voiceprint. For example, the code shows `ref = self.profile.pick_reference(style)` and then `self.tts.update_voiceprint(ref)` before synthesis [1].
- **Modes (Inner/Outer/Coach):** After synthesizing audio, `VoiceCrystal` applies simple psychoacoustic transforms based on mode. In **inner** mode it applies a gentle low-pass filter and lowers volume (`_moving_average` + `inner_volume_scale`) to sound "inside the head"; in **coach** mode it slightly amplifies the audio (`coach_volume_scale`) [2]. These transforms are implemented in `_apply_mode` (see [17]) and invoked for each utterance.
- **Usage:** In the speech loop, `VoiceCrystal` is initialized with a `VoiceMimic` TTS instance and an `AudioIO` player. For example, the reference code shows:

```
self.voice_profile = VoiceProfile(base_dir=profile_dir)
self.voice_profile.load_existing()
self.voice_crystal = VoiceCrystal(
    tts=self.voice, audio_io=self.audio_io, profile=self.voice_profile,
    config=VoiceCrystalConfig(sample_rate=self.config.audio.sample_rate)
)
```

  (from [17] lines 1618–1627). Then calling `voice_crystal.say_outer(text, style=…)` or `say_inner` speaks in the child's cloned voice.

Implementation Reference: The `VoiceCrystal` class is documented as a "High-level voice engine" that chooses style-specific reference samples, runs the TTS, and applies inner/outer/coach transforms [3]. Its `speak()` method (lines 1556–1564 in [17]) picks a reference, updates the voiceprint, synthesizes, and applies the mode. This fulfils the "PPP" (phonemic/prosodic patching) and adaptive voice harvesting with drift monitoring via the `VoiceProfile`.

CrystallineHeart (Emotional Core)

Function: A 1024-node (configurable) lattice of emotional units implementing the core ODE framework. It fuses auditory input into an internal state (arousal, valence, etc.), anneals (cools) over time, and computes global metrics like stress, coherence (GCL), and awareness.

- **State Variables:** Each node has a bit-vector and an emotion scalar. The class stores `self.bits` (array of 0/1), `self.pos` (3D positions), `self.emotion` (float state per node), and a global `temperature` (annealing).
- **Update Step (`step()`):** On each utterance, features (audio loudness, pitch proxy, sentiment) are combined into a stimulus. The method `_update_emotion_field(stimulus)` performs an Euler-like ODE update: `dE/dt = -β*E + α*stimulus + γ*(neighbor_mean-E)` [4]. Then `_anneal()` decays the temperature and occasionally flips some bits stochastically [5]. Finally `_update_metrics()` computes: stress = tanh(local energy), coherence = mean-squared bit alignment, and awareness = coherence·exp(−T) [6]. The step returns a `HeartSnapshot(stress, coherence, awareness)` [7].
- **Metrics (Global Coherence Level):** The GCL is represented by `self.coherence` (range 0–1) after each step. The code shows coherence as the squared mean of bit-values [6]. This metric (along with stress and awareness) is logged and used for gating.

Implementation Reference: The `CrystallineHeart` class is explicitly defined in the sources. The docstring says it implements the lattice with local energy and stress and global coherence [8]. The `_update_metrics()` method computes stress and coherence (see [21] lines 675–684). The `step(...)` function (lines 715–724 in [21]) applies the ODE and returns a snapshot of `(stress, coherence, awareness)`. This fulfills the "1024-node ODE" requirement.

SystemState (Global State Container)

Function: Stores and exposes global state variables each cycle (e.g. coherence/GCL, arousal, etc.) for use by gating and GUI.

- In the provided code, `HeartSnapshot` (stress, coherence, awareness) is used as the state each utterance [7]. One could encapsulate these plus additional arousal/valence into a `SystemState` object.
- **Reference:** The `HeartSnapshot` dataclass (lines 589–598 in [20]) holds `(stress, coherence, awareness)` [9]. These values (especially coherence/GCL and stress) represent the system's internal state. (No separate `SystemState` class appears in the sources; instead the heart snapshot serves this role.)

Note: There is no explicit `SystemState` code in the docs beyond `HeartSnapshot` [9]. Any additional state (e.g. raw arousal) would be integrated similarly.

DRC Gating (Mathematical Control)

Function: Governs when the Deep Reasoning Core (LLM operator) is invoked versus fallback behaviors, based on Global Coherence Level (GCL) and stress. In high-stress or low-GCL ("meltdown") states, LLM usage is throttled off.

- **Gate Computation:** The routing gate R_{LLM} is computed as a sigmoid of a "net stability" score. In code, it normalizes stress and coherence, computes `net_stability = 2*GCL - 1*stress`, then subtracts a strong penalty `3.0 * meltdown_index` (from behavior monitor). Finally it applies a logistic:

```
final = net_stability_score - 3*meltdown_index
gate = 1/(1+exp(-8*(final - 0.5)))
return max(0.01, gate)
```

  (see [9] lines 92–100 and 123–132). This ensures the LLM gate goes low when stress is high or GCL is low [10] [11].
- **SpeechLoop Usage:** In each utterance, after updating the heart and behavior, the code does:

```
llm_gate = compute_llm_gate(metrics, meltdown_index)
self.heart.adjust_temperature_for_meltdown(meltdown_index)
if llm_gate > CONFIG.LLM_GATE_THRESHOLD:
    corrected = self.llm_operator.run(raw_text, metrics)
else:
    corrected = self.deterministic_corrector.process(raw_text)
    if meltdown_index > CONFIG.MELTDOWN_THRESHOLD:
        self.calming.play_calming("meltdown_script", first_person=True)
self.speak_response(corrected, style=...)
```

  (see [10] lines 177–185 and 190–198). In other words, when `llm_gate` is high, the LLM "rich semantic" operator is used; when it is low, a grammar-corrector or calm fallback runs.

Implementation Reference: The sigmoid-based gate function is given in [9] (as Python code). The updated `SpeechLoop._process_utterance()` pseudocode in [10] shows exactly how the gate is applied (lines 179–187 and 190–197). This matches the specification of a GCL-governed throttle with autopoietic fallbacks.

InnerVoiceEngine (First-Person Self-Talk)

Function: When a stress event is detected (anxiety, perseveration, etc.), this module generates a short *first-person* supportive line (e.g. "I can take a breath") and plays it softly through the child's own voice. It reinforces the idea of an internal calming dialogue.

- **Templates:** A set of short first-person phrases is defined for each event (e.g. the "anxious" event has "I can slow down and take a gentle breath..." etc.).

- **Acoustic Styling:** The line is played via `VoiceCrystal.say_inner(...)`, which uses the child's cloned voice but applies the **inner** mode acoustic transform (quieter, slightly muffled).
- **Logging:** Each inner-voice utterance is logged (as a "guidance event").

Implementation Reference: The `InnerVoiceEngine` class in [15] (lines 2757–2765) encapsulates this logic. Its `reinforce_for_event(event, last_text, corrected_text, prosody_wav, sr)` builds an inner-text (`_build_inner_text()`) and calls `self.voice.say_inner(text, style)` [12] [13]. For example, on an "anxious" event it might speak "I can slow down and breathe…" in a calm style. The code explicitly calls `self.voice.say_inner(…)`, ensuring the audio is first-person and softly rendered [12] [13].

PhenotypingTracker (Vocal Fragment Classification)

Function (desired): Continuously classify incoming audio fragments as **clear speech**, **ambiguous speech**, or **non-verbal** (e.g. humming, sighs, cries) and annotate them for further processing or caregiver info. Ideally this would label each utterance with its "phenotype" in real time.

Note: We did *not* find any explicit implementation of a phenotyping tracker in the provided sources. The closest related code is the enhanced VAD (`is_actual_speech`) which filters out non-speech sounds [14]. A production implementation would likely hook into the speech-processing pipeline (using RMS, zero-crossings, spectral features) to tag fragments, but this functionality would need to be added. (Because no code was found, we note it as a needed module that should integrate with the `SpeechLoop` and GUI to display classifications.)

SpeechLoop (Main Audio Pipeline)

Function: The real-time processing loop that ties everything together. It continuously captures audio, applies VAD, runs ASR, processes the text, updates emotion, makes decisions, and produces output audio (correction or guidance).

- **VAD/ASR:** Each audio chunk is passed to a neurodiversity-tuned VAD. If voice activity is detected (see the provided `is_actual_speech` logic [14]), the chunk is sent to an offline ASR engine (e.g. Whisper).
- **Speech Correction:** The ASR text goes through a speech processor (grammar/phrase matching). Any corrected text is then logged as practice data.
- **Emotion Update:** The raw audio features and/or text sentiment are fed into the `CrystallineHeart` via something like `heart_snapshot = HEART.step(audio_rms, audio_pitch, sentiment)`. (The helper `update_heart_from_voice()` in [21] shows extracting RMS, pitch proxy, and simple sentiment, then calling `HEART.step()`) [15] [7].
- **Gating & Decision:** Using the new heart metrics (`stress`, `coherence`) and the behavior monitor's meltdown index, the loop computes gates as above. It updates the heart's annealing temperature via `heart.adjust_temperature_for_meltdown(meltdown)`. Then it chooses an operator: e.g. apply the LLM to do rich expansion if `llm_gate` is high, otherwise do a deterministic correction. If in crisis it may trigger a calming script (play via `InnerVoiceEngine` or a coach voice).

- **Output:** The final text (corrected or generated) is spoken back. Non-problematic repeats use `VoiceCrystal.say_outer(corrected_text, style)` to mimic the child's voice in second person. Calming prompts use `say_coach` or the inner-voice mechanism as needed.
- **Reference:** The pseudocode in [10] summarizes these steps. For example, it shows updating heart and behavior, extracting metrics, computing gates, and finally speaking response via `self.speak_response(corrected, style=…)`. Also, the initial part of that code demonstrates updating the heart from the raw audio/text. Additionally, [17] (lines 1618–1627) shows how the loop initializes its components (`AudioIO`, `VoiceMimic`, `VoiceCrystal`, etc.).

GUI (KivyMD Interface)

Function: A dual-view interface – one "Molly Dashboard" for the caregiver and one avatar view for the child (Jackson) – showing real-time metrics, stress/coherence charts, phenotyping results, and a friendly avatar.

Implementation Notes: The provided sources include plans for a WebSocket-based avatar and charts (in the "Merged system code" PDF), but no final KivyMD code is shown. A production implementation would use KivyMD to build two screens: - The *Molly Dashboard* displays telemetry (stress, GCL, meltdown index), gate levels, and logs of events (phenotype annotations, inner-voice usage). - The *Jackson Avatar* view shows a simple character or avatar, lip-syncing or animating to the speech output. Because no concrete code was found, we simply note that the GUI should pull from the same telemetry store. All GUI calls must be thread-safe (Kivy has its own thread) and updated at ~10 Hz or slower to keep latency low.

Packaging (Desktop & Mobile)

- **Desktop (PyInstaller):** We include an installer script that bundles the Python environment and all modules into a single executable. For example, the provided `installer/build_all.py` uses PyInstaller's `--onefile` mode to package `echo_server.py` into a desktop binary ("EchoServerBundle") [16] [17]. This ensures the entire app (including model files and Kivy assets) runs offline. Icons and entry scripts are added via PyInstaller's `--icon` and `--add-data` options as needed.
- **Android (Buildozer):** For mobile, the approach is to use Kivy + Python-for-Android. The docs show installing Buildozer and SDK tools (`pip install buildozer; sudo apt install openjdk-17-jdk android-sdk`), then creating a minimal Kivy launcher app (`mobile/main.py`) and running `buildozer android debug` to produce an APK [18] [19]. Buildozer's `buildozer.spec` would list all Python requirements (Whisper, TTS, numpy, KivyMD, etc.) and any Java permissions. The child's device runs the same code and models fully offline.

Finally, all modules must be connected: the main script wires together `SpeechLoop`, `CrystallineHeart`, `VoiceCrystal`, `InnerVoiceEngine`, etc., ensuring that each voice event flows through every component. Latency is minimized by using efficient libraries (e.g. int8 Whisper, coqui-TTS), and care is taken with thread-safety (e.g. Kivy updates on the main thread, audio I/O in background threads).

Sources: The above design is based on the provided Echo v4 integration guides and code snippets [3] [6] [7] [12] [16] [18], which illustrate the implementations of `VoiceCrystal`, `CrystallineHeart`, gating, inner voice, and packaging. Components for which no explicit source code was found (e.g. a "PhenotypingTracker" or the full KivyMD GUI) would need to be developed following the same principles.

---

1 2 3 12 13 System overview explanation-1.pdf
file://file-QbbgFbK62FpjpH8PMPEagM

4 5 6 7 8 9 10 11 15 autism - Merged system code.pdf
file://file-8ag7FfDoPvMB4z2nWAf3T5

14 18 19 autism Project purpose.txt
file://file-7srqR54YZshguhuGrXjwb7

16 17 autism - Echo v4 setup guide.pdf
file://file-6QRNCpWdg2zoFQuxgC5xjF