# Multi-Agent Swarm Syntax: An Approach to Live Coding with AI

Sven Hollowell
University of Bristol
Bristol, United Kingdom
sven.hollowell@bristol.ac.uk

Pete Bennett
University of Bristol
Bristol, United Kingdom
pete.bennett@bristol.ac.uk

Paul Marshall
University of Bristol
Bristol, United Kingdom
p.marshall@bristol.ac.uk

## Abstract

Collaborative live coding treats the shared text buffer as a stage where the act of typing forms part of the performance. Standard AI assistants, however, prioritise efficiency and opacity, often inserting code instantaneously and reducing the generative process to an invisible operation. This conflicts with live coding's emphasis on visibility and audience legibility.

We present a system built on top of Strudel, an interactive web-based live-coding environment, that re-frames Large Language Models (LLMs) as visible co-performers. Agents are spawned through inline comments (e.g., `/// chaotic bass agent`) and edit code through distinct cursors, generating text character-by-character alongside the human performer. A Conflict-Free Replicated Data Type (CRDT) editing model allows concurrent human and agent edits while maintaining a consistent shared document state.

The system was evaluated through a collaborative autoethnographic study. The authors engaged in iterative performance sessions to identify emerging interaction patterns. Observations suggest that visible machine participation shifts the performer from direct creation toward curation, while preserving the performative "liveness" of live coding.

## CCS Concepts

• **Applied computing** → **Sound and music computing**; • **Computing methodologies** → *Artificial intelligence.*

## Keywords

Live Coding, Large Language Models, Agency, Co-Performance, Strudel, CRDTs

**ACM Reference Format:**
Sven Hollowell, Pete Bennett, and Paul Marshall. 2026. Multi-Agent Swarm Syntax: An Approach to Live Coding with AI. In *Proceedings of International Conference on New Interfaces for Musical Expression (NIME '26).* ACM, New York, NY, USA, 5 pages.

## 1 Introduction

Live coding is a performance practice in which code functions simultaneously as instrument, notation, and performance surface. A defining characteristic of the practice is the exposure of process: audiences observe the construction of music in real time, often summarised by the principle of "show your screen" [2, 14]. The act of writing code is therefore not only a means of composing music but part of the performance itself [2].

While starting a performance from a "blank slate" can be creatively desirable (since the emergence of structure is fully visible), it introduces significant performative inertia, requiring performers to simultaneously manage syntax, structure, and sound production before musical material emerges [6, 8].

Standard AI code generators are framed as productivity tools, often inserting large blocks of code instantaneously. This opaque interaction model directly conflicts with live coding's emphasis on temporal process and visible authorship. To address this, we present StrudelMASS[1], a system that re-frames Large Language Models (LLMs) as visible co-performers.

## Contributions

This paper makes four main contributions:

- **A Multi-Agent Interface for Embodied Co-Performance:** We introduce a novel interface paradigm where AI agents inhabit the shared text buffer as visible, distinct actors. By generating code through simulated, character-by-character typing via independent cursors, the system preserves the performative visibility and "narrative of making" central to live coding.
- **A Conflict-Free Infrastructure for Hybrid Live Coding:** We present a technical implementation utilizing **Conflict-Free Replicated Data Types (CRDTs)** to enable simultaneous, non-blocking editing by multiple agents and human performers. This architecture ensures eventual consistency of the document state even during moments of high-frequency editing.
- **Syntactic Reliability via Domain-Specific Retrieval:** We demonstrate a **Retrieval-Augmented Generation (RAG)** pipeline tailored for the Strudel language. By dynamically injecting relevant, validated syntax examples into the agent's context window during generation, we enable the use of lower-latency, cost-effective models while significantly reducing hallucination rates and syntax errors.
- **Adversarial Agency as a Creative Design Pattern:** We propose an interaction framework that moves beyond the "assistant" model, positioning AI agents as semi-autonomous co-performers capable of conflicting with, overwriting, or subverting human intent. We argue that this "productive friction" serves as a generative driver in live performance, distinct from the optimization-focused goals of standard software engineering assistants.

## 2 Background & Related Work

### 2.1 Live Coding & Networked Environments

Live coding has been described as a practice where code operates as both representation and action, collapsing composition and performance into a single activity [2]. A defining necessity of the practice is legible process [14], which introduces tension when automated generation obscures performance.

[1] Demo and code available at https://strudelmass.sven.zone and https://codeberg.org/supersational/StrudelMASS


*NIME '26, June 23–26, 2026, London, UK*

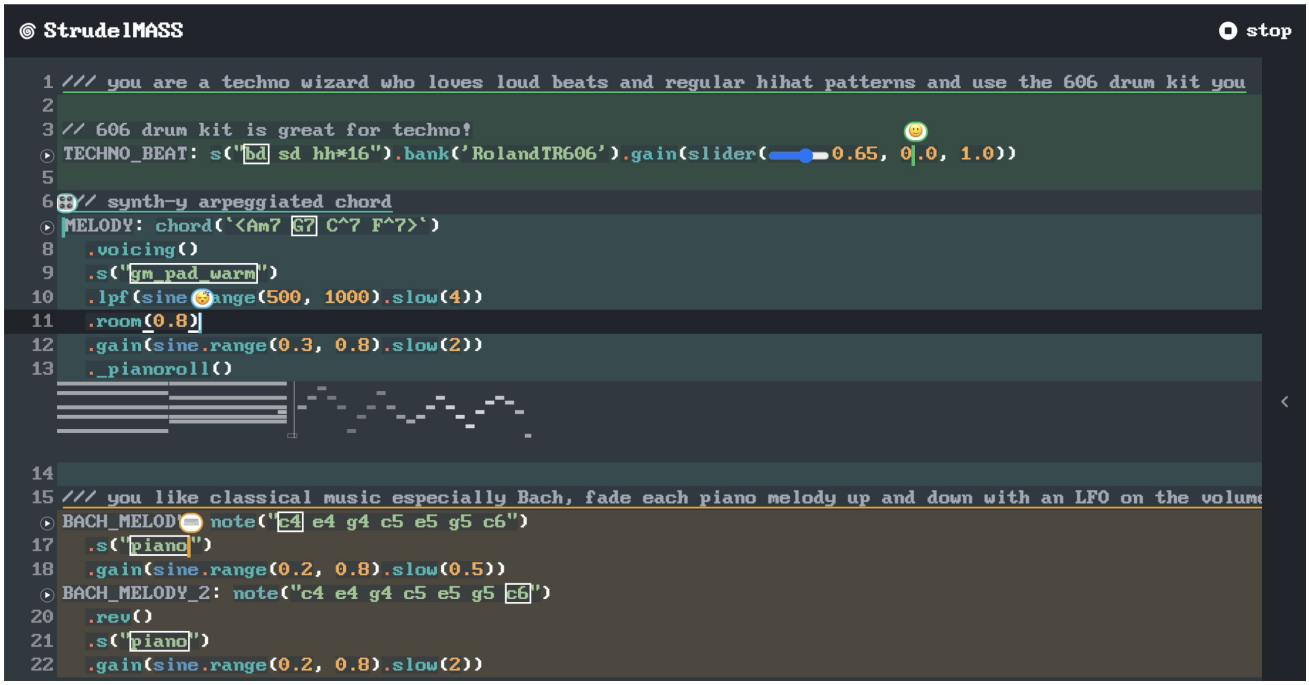


**Figure 1: Multi-agent editing in StrudelMASS: agents defined through `/// agent` directives appear as visible cursors within their coloured buffer 'zone'. Some agents are currently typing, while other agents remain visible in the editor.**

The evolution of collaborative, web-based live coding—from early browser environments like Lich.js [7] and Gibber [10] to contemporary platforms like Strudel [13]—has increasingly centered the shared text buffer as a site for real-time coordination. This culminates in shared-authorship experiments like Pastagang [9], where participants engage in continuous, visible overwriting without fixed ownership.

## 2.2 AI Generation & Machine Agency

Algorithmic generation has long formed part of live coding practice. While early visual systems like ixi lang's Betablocker [5] and recent spatial agent systems [1, 11, 12] explore autonomous, non-human actors in shared spaces, contemporary AI code generation (e.g., Autopia [4], Tidal-MerzA [15], VibeJam [3]) is typically framed as assistive and cooperative, supporting performer goals through optimization.

To analyze our shift away from this cooperative model, we apply Xambó and Roma's [16] framework of machine agency, using their four dimensions as an analytical lens:

- **Legibility:** Does character-by-character generation transform an opaque "black box" into a visible co-performer for both the audience and the artist?
- **Modifiability:** How can performers 'negotiate' with an AI that occupies and edits the same shared text-buffer?
- **Predictability:** As Xambó notes, an agent behaving unpredictably conveys a stronger "impression of having a distinct agency" [16]. In this context, how can performer mastery shift from absolute control to a reactive "steering" of the agent's unexpected output?
- **Cardinality:** What are the limits of a performer's cognitive and visual bandwidth in multi-agent environments? Can the ability to scale an agent swarm be utilized as a dynamic strategy for performance?

# 3 Designing StrudelMASS

We built upon Strudel [13] to develop interaction mechanisms that make multi-agent activity legible, bounded, and performable.

## 3.1 Agentic Cursors & Bounded Zones

Each agent is represented as a cursor within the editor, generating edits character-by-character to allow performers and audiences to observe changes emerging over time. Agents are defined by inline comments (e.g., `/// you love breakcore`). To prevent whole-document takeover, agents are confined to bounded text *zones* between directive lines; human performers can edit anywhere, but agents cannot escape their designated region (Fig. 1).

## 3.2 CRDT-Based Editing & History

Concurrent editing is implemented using a Conflict-Free Replicated Data Type (CRDT) model, guaranteeing convergence of text operations across multiple editors. To provide situational awareness, agents are fed a short-term memory of recent edits made to the document (by both humans and agents), utilizing a recency-focused window that nudges continuity without enabling long-term planning.

## 3.3 Retrieval-Augmented Syntax Support

To enable the use of low-latency models (`gemini-2.5-flash-lite`) while maintaining syntactic reliability, we incorporate a retrieval-augmented generation (RAG) layer. The system dynamically retrieves short, curated Strudel syntax examples based on the agent's current intent and local code context, injecting them into the prompt. This reduces latency and model cost by grounding smaller models in known, valid syntax patterns.

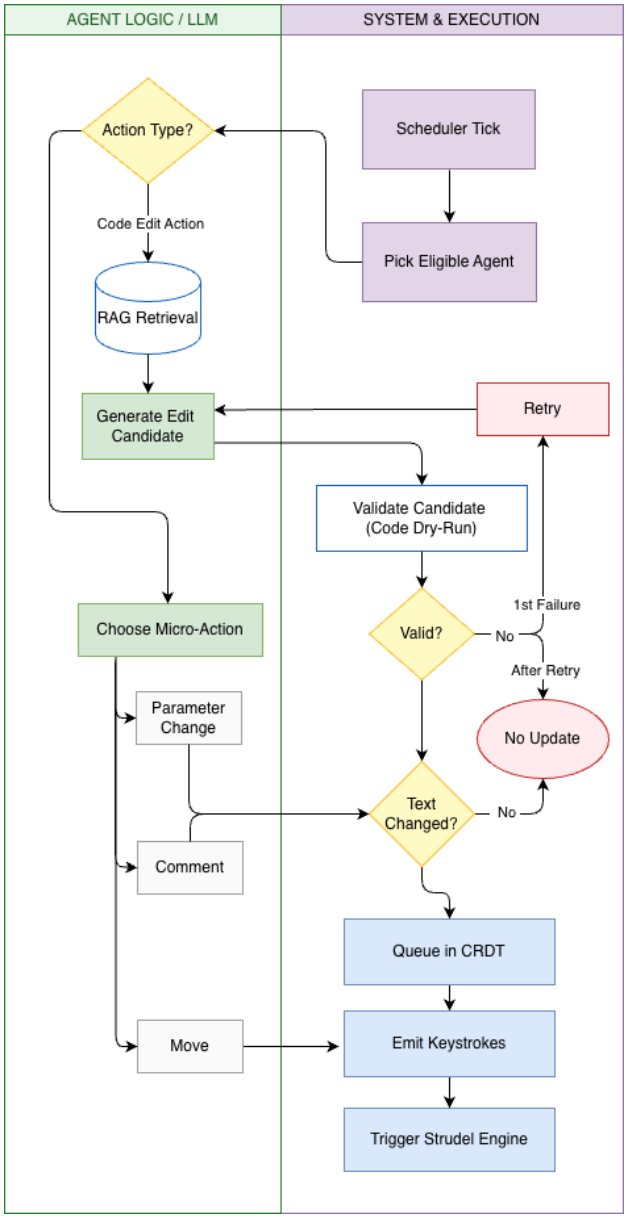


**Figure 2: Agent edit pipeline showing how autonomous edits are selected, validated, and applied within the shared editor.**

### 3.4 Zones and Areas of Control

To prevent whole-document takeover, agents are strictly confined to dynamically created text *zones* (spanning from their inline directive to the next). Any agent edit attempting to escape these boundaries is rejected. This grants agents local autonomy while preserving visual attribution and reserving the un-zoned top level of the document as a safe, human-only space.

### 3.5 Safety and Constraints

To maintain performance continuity, the system enforces the following constraints (Fig. 2):

- **Validation & Fail-Soft Execution:** Agent edits are parsed before acceptance. Because concurrent editing can introduce transient syntax errors, invalid patterns are dropped from the runtime state.
- **Execution Control:** Initial buffer evaluation remains strictly performer-triggered; subsequent pattern updates may be initiated by either the human or active agents.
- **Lifecycle Control:** Agents can be instantly paused, disabled, or removed by the performer at any time by modifying their directive line.

## 4 Playing StrudelMASS

To illustrate the system in practice, we present a composite vignette synthesized from recurring interaction patterns observed across several autoethnographic self-study sessions.

### 4.1 The Summoning

The performance begins with an empty buffer. To initiate a musical idea, the performer types a directive. As soon as the final forward slash is typed in the trigger line, the system recognizes the syntax:

```
/// airy synthwave melody
```

Immediate visual feedback confirms the system's recognition: the command line is underlined in a distinct color, and a new cursor appears adjacent to the text. Initially, this cursor displays a "sleeping" icon, indicating that the agent is instantiated but waiting for the prompt to be completed. Upon execution, the cursor state shifts to a "thinking" icon, signaling the latency period of token generation.

### 4.2 Visible Generation

Moments later, the cursor state changes to a "keyboard" icon, and the agent begins to generate code. Unlike standard autocomplete, the code is not dumped instantaneously; it streams character-by-character, simulating the presence of a typist. The following block emerges in the editor:

```
1
2 /// airy synthwave melody
3 MELODY: note("c4 e4 g4 b4 d5 g4 b4 5")
4   .s('gm_pad_warm')
5   .lpf(sine.range(500, 2000))
6   .room(0.7)
7
```

**Figure 3: Agent code generation following instantiation.**

Because StrudelMASS evaluates code blocks on update, the pattern is immediately activated. A warm, oscillating synthesiser pad fills the silence. An audience can see both the source of the code (the agent's distinct cursor) and the resulting sound simultaneously.

### 4.3 The Edit War

While the generated melody fits the genre, the performer finds the timbre too aggressive; the low-pass filter (LPF) sweep up to 2000Hz creates a sharp resonance that clashes with the intended mood. The performer intervenes, moving their own cursor to the line and manually editing the upper limit down to 800Hz:

*User Edit:* `.lpf(sine.range(500, 800))`

The agent, however, is still active and monitoring the state of its zone. Adhering to its "airy" persona instruction (which implies high-frequency content), the agent detects the deviation. It moves its cursor back to the parameter and overwrites the user's edit, pushing the value even higher to emphasize the airiness:

*Agent Re-Edit:* `.lpf(sine.range(500, 4000))`

This moment represents a critical divergence from standard assistive AI. The system has rejected the human's correction, forcing the performer into a negotiation. They must now choose to fight for the lower frequency, accept the agent's brighter texture, modify the agent's prompt (e.g., changing "airy" to "subdued"), to align their goals, or add a short inline comment addressed to the agent (e.g., `// please keep it darker and low-end`) to steer subsequent edits—without changing the prompt itself.

## 4.4 Shared Control: Melody and Synthesis

The performer introduces a bass pattern manually, establishing the melodic material while keeping the synthesis simple:

```
BASS: note("c2 ~ c2 g1")
  .s('gm_electric_bass_finger')
  .lpf(900)
```

Rather than requesting melodic variation, the agent is instructed to shape the sound:

```
/// make the bass more animated and evolving
```

The agent begins modifying synthesis parameters instead of pitch content, introducing motion through filter and spatial changes:

```
.lpf(sine.range(400, 1800))
.room(0.5)
```

The performer retains control of the melodic line, editing the note pattern in response to the changing texture:

```
note("c2 ~ d2 g1")
```

and later:

```
note("c2 ~ bb2 g2")
```

To gain finer control over the evolving brightness, the performer introduces a simple GUI control during performance:

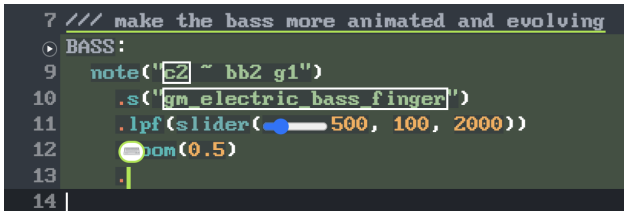


**Figure 4: Adding a `.slider()` control.**

The slider exposes a continuously adjustable parameter that can be manipulated to modify the code parameters. While the agent continues modifying synthesis behaviour around the filter and effects, the performer moves the slider to control overall energy and spectral density in real time.

# 5 Discussion

Across our self-study sessions, agent activity was experienced less as "automation" and more as a stream of new events in time: edits arrived continuously, and the performer's primary work shifted toward responding, steering, and occasionally resisting those edits. Musical direction therefore emerged through short, asynchronous, reaction cycles, rather than from a single authorial plan executed linearly.

We evaluate the implications of this shift across four key themes.

## 5.1 Shifting Roles: Accessibility, Curation, and Cognitive Overload

Observation: Agents reduced the empty buffer's initial inertia by introducing immediate material. In practice, this replaced "starting from nothing" with a mode of selective response.

Interpretation: Because agents are defined through natural-language directives, performers can also instantiate narrowly constrained behaviours (e.g., an agent instructed only to fix missing brackets and correct syntax errors). This suggests a possible accessibility direction—for example, as a tool for musicians for whom rapid, continuous typing is physically prohibitive—but we have not evaluated accessibility outcomes here.

Implication: This reframing is not uniformly "easier." Offloading generation shifts the performer from a micro-level typist to a macro-level curator: monitoring multiple cursors, interpreting intent from directives, and deciding when to intervene or disable agents. During the sessions, this managerial role sometimes increased cognitive load in precisely the moments where live coding is already attention-limited (e.g., when multiple agents edited simultaneously).

## 5.2 Liveness and the Authentic Co-Performer

Observation: Agent edits were perceived as performance actions because they unfolded character-by-character and were attributable to a persistent cursor and directive line.

Interpretation: Liveness was maintained not by correctness or speed, but by preserving interpretable process: seeing what changed, who changed it, and how it unfolded over time.

Implication: Treating the agent as an explicit actor (rather than an invisible layer) reframes AI support as co-performance, where the machine's labour becomes part of what is shown.

## 5.3 Productive Friction and Negotiated Authorship

Observation: In the "edit war" vignette, both performer and agent targeted the same parameter (LPF cutoff), producing repeated overwrite cycles.

Interpretation: Interaction stabilised when responsibility separated across dimensions (human-controlled pitch vs. agent-controlled timbre), allowing negotiation to occur through constraint-setting, and removing the need to "win" discrete edits.

Implication: When an agent's stylistic goals diverge from the performer's, the resulting friction can become productive: it interrupts habitual trajectories and turns authorship into an ongoing negotiation.

## 5.4 System Fragility and Failure Modes

Observation: Several failure modes were common in practice. Because agents do not hear audio, they may combine parameters in musically counterproductive ways (e.g., reducing gain toward silence or over-processing with effects). Agents' reliance on buffer history often triggered runaway repetition (reusing fragments) or accretive density (adding notes).

Interpretation: These bugs reflect the limits of text-only context for musical decision-making. We experimented with providing audio as an input signal, but in our tests models used it mainly as a coarse cue for loudness or "busyness", with little impact on musical quality.

Implication: These failure modes clarify where agency is currently brittle and motivates future work on feedback signals,

constraint design, and mechanisms for musically meaningful self-correction.

## 6 Conclusion

We presented Multi-Agent Swarm Syntax, a live coding system that reframes AI models as visible co-performers rather than hidden assistants. By combining multi-cursor interaction with CRDT-based editing and constrained agent autonomy, the system allows human and machine contributions to coexist within the same performance space.

The work suggests that exposing machine activity, rather than hiding it, may better align AI systems with live coding's emphasis on process and liveness. Future work will explore broader performer adoption, temporal reasoning for agents, and alternative forms of human–AI negotiation in real-time creative systems.

## 7 Ethics Statement

This study aligns with University of Bristol ethical and data protection guidelines. Research data is derived exclusively from the author's autoethnographic self-study.

## 8 Acknowledgements

We are deeply grateful to the Strudel team for open-sourcing their platform. This project represents an independent, academic research fork and is not affiliated with, or endorsed by, the original Strudel developers.

Sven Hollowell is supported by the UKRI Centre for Doctoral Training in Interactive Artificial Intelligence (EP/S022937/1).

## References

[1] Jack Armitage, Victor Shepardson, and Thor Magnusson. 2024. Tölvera: Composing with basal agencies. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 282–291.

[2] Nick Collins. 2003. Live coding in laptop performance. In *Proceedings of the International Computer Music Conference*. 1–4.

[3] Nicolas Jonason. 2026. *vibejam*. https://github.com/erl-j/vibejam

[4] Norah Lorway, Edward J Powley, Arthur Wilson, John A Speakman, and Matthew Jarvis. 2019. Autopia: An AI Collaborator for Live Coding Music Performances. In *Proceedings of the International Conference on Live Coding (ICLC)*. ICLC, Madrid, Spain, 1–8.

[5] Thor Magnusson. 2011. ixi lang: a SuperCollider parasite for live coding. (2011).

[6] Thor Magnusson. 2014. Herding cats: Observing live coding in the wild. *Computer Music Journal* 38, 1 (2014), 8–16.

[7] Chad McKinney. 2014. Quick live coding collaboration in the web browser. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 379–382.

[8] Click Nilson. 2007. Live coding practice. In *Proceedings of the 7th international conference on New interfaces for musical expression*. ACM, New York, NY, USA, 112–117.

[9] Pastagang. 2025. Pastagang: Jamming together far apart. https://doc.patternclub.org/s/tBub4TxF1. In *Proceedings of Alpaca 2025: Algorithmic Patterns in the Creative Arts*. Online. doi:10.5281/zenodo.17084430 Published in Alpaca 2025 proceedings. Accessed 2026-02-13.

[10] Charlie Roberts. 2022. Rethinking the laptop ensemble: Networked live coding with Gibber. *Computer Music Journal* 46, 2 (2022), 1–15.

[11] Gerard Roma. 2023. Agent-based music live coding: sonic adventures in 2D. *Organised Sound* 28, 2 (2023), 231–240.

[12] Gerard Roma. 2025. Live coding sonic flocks. In *2025 International Conference on Live Coding*.

[13] Felix Roos and Alex McLean. 2023. Strudel: Live Coding Patterns on the Web. In *Proceedings of the 7th International Conference on Live Coding*. 1–8.

[14] Adrian Ward, Julian Rohrhuber, Fredrik Olofsson, Alex McLean, Dave Griffiths, Nick Collins, and Amy Alexander. 2004. Live algorithm programming and the temporary organisation of music. In *Proceedings of the International Computer Music Conference*. 1–4.

[15] Elizabeth Wilson, George Fazekas, and Geraint Wiggins. 2024. Tidal MerzA: Combining affective modelling and autonomous code generation through Reinforcement Learning. In *Proceedings of the International Conference on AI and Music Creativity (AIMC)*. AIMC, Oxford, UK, 1–10. https://aimc2024.pubpub.org/pub/4na79l3

[16] Anna Xambó and Gerard Roma. 2024. Human–machine agencies in live coding for music performance. *Journal of New Music Research* 53, 1 (2024), 1–17.