International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Parallel Editing in p5.js and Paper.js with ShareDB: An OT-based Architecture and a CRDT Metadata Lane

Ravi Dayani

ravip1152@gmail.com

Abstract:
Real-time co-authoring has become a first-class requirement for creative coding and vector graphics on the web. This paper presents a reference architecture for parallel editing across p5.js (immediate-mode) and Paper.js (retained-mode), built on ShareDB's Operational Transformation (OT) for structured JSON documents [1][4][6][16]. We define a unified schema and semantic, fine-grained operations (e.g., path.segment.update, parameter state.set) that preserve user intent under concurrency, along with multi-user undo/redo and workspace awareness cues to improve collaboration quality [16][19]. To strengthen offline behavior and reduce contention, we add a CRDT metadata lane (e.g., Yjs) for annotations/comments, merging deterministically without server ordering and complementing the OT core [13][14]. Synthetic evaluation shows low end-to-end latencies with immediate local echo, sub-millisecond transform/apply costs for most ops, high intention preservation on concurrent vector edits, and robust offline resilience for annotations. We conclude with practical guidance—hierarchical addressing, batching, stable IDs—and discuss trade-offs between immediate- and retained-mode collaboration surfaces.

Keywords: Real-time collaboration, Operational Transformation, CRDT; p5.js, Paper.js, ShareDB, Yjs, scene graph, parallel editing, workspace awareness, selective undo, WebSocket.

I. INTRODUCTION

Creative coding platforms such as p5.js and Paper.js have become widely used by artists, designers, and educators for rapidly prototyping visual ideas in the browser [4–6]. As creative work increasingly shifts toward online and collaborative environments, parallel editing—multiple participants co-authoring the same sketch or vector composition in real time—has emerged as both a technical challenge and a promising extension of existing creative practices. Unlike turn-taking or file-locking models, true parallel editing requires that users' actions propagate with low latency, remain consistent across clients, and avoid destructive conflicts [1][16][19].

This paper investigates how to enable such collaboration for p5.js and Paper.js using ShareDB, a real-time synchronization engine based on Operational Transformation (OT) [1–3][16]. While OT is well established in collaborative text editing, applying similar guarantees to graphics-centric, event-driven environments introduces unique challenges. p5.js uses an immediate-mode rendering model in which each frame redraws from program state, whereas Paper.js maintains a retained-mode scene graph with mutable vector objects [4–6]. These contrasting paradigms require different collaboration strategies: p5.js emphasizes synchronization of code and state variables that drive rendering, while Paper.js centers on shared, structured operations on a hierarchical vector model.

Using ShareDB as the synchronization backbone, we explore how to (1) represent artwork and code changes as composable operations, (2) map user interactions onto a unified document model, and (3)

maintain consistency and user intent during concurrent edits [1][2][16]. The effectiveness of OT in these contexts depends heavily on document structure and operation granularity. Coarse operations (e.g., replacing an entire sketch) minimize conflicts but hinder collaboration [8][16], while fine-grained edits (e.g., modifying a path segment or adjusting a parameter) improve parallelism at the cost of greater transform complexity [4][16][17]. For Paper.js in particular, hierarchical addressing (project → layer → item → segment) enables conflict-aware transforms when users edit neighboring parts of the same vector object [6][16].

Alongside technical concerns, we consider the human factors essential to a productive collaborative experience: real-time awareness cues (cursors, selection highlights) [19], recoverability mechanisms such as multi-user undo/redo [16][18], and fairness policies that prevent silent overwrites [13][14]. Performance constraints—including latency under variable network conditions, high-frequency edit bursts, and contention around shared objects—further shape design decisions for batching, indexing, and operation schemas [1][12][17]. Ultimately, the perceived quality of collaboration depends on responsive local feedback, intelligible conflict resolution, and predictable behavior during disconnection and reconnection [8][14][19].

Contributions

This work presents:

- A reference architecture for real-time collaborative editing in p5.js and Paper.js built on ShareDB, including data flows, operation schemas, and OT strategies tailored to immediate- vs. retained-mode rendering [1–6].
- Design patterns for translating user interactions into OT operations for code edits, parameter updates, and vector-editing primitives [4][6][16][17].
- A cross-framework evaluation comparing collaboration metrics (latency, transform cost, conflict rate) and user experience indicators (awareness, recoverability, task performance) [1][12][16][17][19].
- Guidelines for multi-user undo/redo, access control, and offline resilience, including recommendations for logging, causality tracking, and conflict visualization [1][14][16][18].

Research Questions

- RQ1: How should shared state be modeled to support low-conflict, high-parallelism editing across immediate- and retained-mode graphics engines? [4][6][16]
- RQ2: What operation granularity and addressing schemes best preserve user intent while minimizing transform complexity? [6][16][17]
- RQ3: Which awareness and recoverability mechanisms most enhance collaborative experience without degrading performance? [18][19]
- RQ4: How do network conditions and workload patterns influence latency, conflict frequency, and perceived smoothness of collaboration? [1][12][17]

Scope

We focus on browser-based collaboration using Node.js, WebSockets, and ShareDB's OT core [1–3]. While OT is our primary mechanism, we briefly compare it with CRDTs to contextualize trade-offs for real-time graphics workloads [13–15].

By integrating OT with domain-specific data models for p5.js and Paper.js, this work aims to make real-time, multi-user visual composition both technically robust and creatively accessible.

II. RELATED WORK

A. *Foundations of real-time collaborative editing*

Operational Transformation (OT) and Conflict-Free Replicated Data Types (CRDTs) remain the two primary approaches to concurrent editing [13][16]. OT research established guarantees such as convergence, causality, and intention preservation, along with correctness frameworks and optimizations for reducing transform cost over long histories [16][17]. CRDTs provide strong eventual consistency without central coordination, supported by extensive surveys and practical libraries for sequences, trees, and JSON-like structures [13][14][15]. Recent implementations offer editor-oriented features such as shared cursors, offline workflows, and snapshotting [14][15].

B. *Web frameworks for shared state and editing*

ShareDB is a widely adopted OT-based backend for JSON documents, offering WebSockets, presence, queries, and offline resynchronization—capabilities relevant to granular multi-user graphics editing [1][2][3]. On the CRDT side, Yjs supplies high-performance shared types and providers for transport and storage, enabling collaborative code and document editors without a single authoritative server [14][15]. CodeMirror 6 demonstrates a contrasting centralized OT model and can also integrate with Yjs, mirroring architectures used in p5.js-based web IDEs [4][8].

C. *Collaborative graphics editors (retained-mode)*

Production tools like Figma document custom real-time architectures tailored to structured vector scene graphs rather than text-style OT pipelines [9]. Excalidraw shows how retained-mode graphics can be synchronized using CRDTs such as Yjs over WebRTC or Socket.IO [10][14][15]. Complementary work on vector representations and performance, including DeepSVG and SSVG, informs data modeling and low-latency rendering strategies relevant to collaborative vector editing [11][12].

D. *Graphics frameworks: immediate vs. retained mode*

p5.js uses immediate-mode rendering, making shared code and state variables the primary collaboration surface [4]. Paper.js provides a retained-mode scene graph (Project → Layers → Items → Paths), supporting structured operations on paths and segments that map naturally to addressable OT/CRDT edits [5][6][7].

E. *Awareness, presence, and recoverability*

CSCW research highlights workspace awareness—cursors, selections, viewports—as central to fluid multi-user interaction [19]. Multi-user undo/redo has been extensively studied within OT systems, with selective-undo frameworks ensuring convergence and intention preservation even under interleaved edits [16][18].

F. *Positioning of our work*

Existing work provides mature synchronization engines (ShareDB, Yjs), production-grade collaborative editors (Figma, Excalidraw), and strong theoretical foundations for consistency, awareness, and undo/redo [1][3][9][10][13]–[15][19]. However, guidance is limited on operation schemas that bridge immediate-mode (p5.js) and retained-mode (Paper.js) graphics within ShareDB's JSON-OT model, including address spaces for paths/segments and transformation rules for vector-editing primitives [1][2][4]–[7][16][17]. Our work contributes a unified modeling and transformation approach tailored to these demands.

III. METHODS

A. *System architecture and runtime*

Our system enables parallel editing for both p5.js (immediate-mode, rasterized rendering) and Paper.js (retained-mode, vector scene graph) by synchronizing structured JSON documents over ShareDB using Operational Transformation (OT) [1][3][16]. Clients connect via WebSockets, publish granular operations, and subscribe to live updates; the server validates and transforms operations against concurrent ones and then commits them to the canonical document [1][2][16].

Rationale. p5.js recomputes the frame from program state on each draw() call, so collaboration focuses on code/state deltas (parameters, variables, assets) [4]. Paper.js exposes a scene graph (Project → Layer → Item → Path → Segment), so collaboration centers on structurally addressable edits to items and segments [5][6].

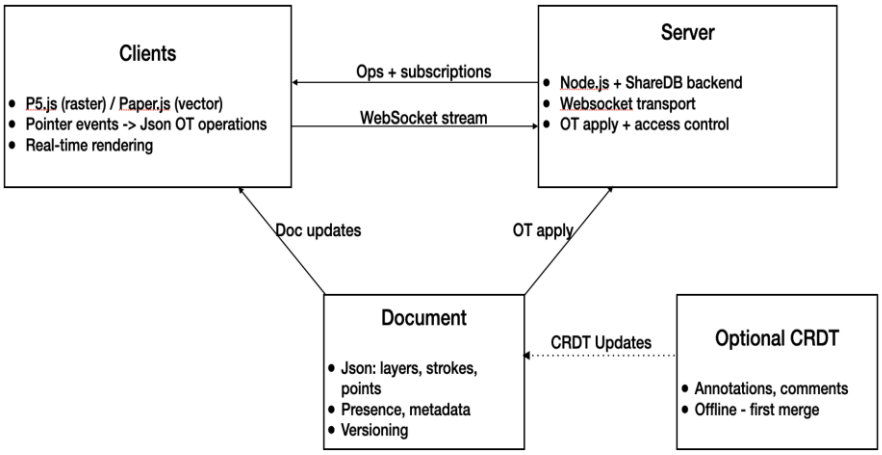

Fig 1. End-to-end architecture (clients, server, document, optional CRDT path)

Runtime components

- *Clients:* Browser apps built with p5.js or Paper.js. We instrument pointer/keyboard events and editor actions, map them to JSON OT operations, and render local echo immediately [4][6].
- *Server*: Node.js + ShareDB, exposing WebSocket endpoints, enforcing access control, and running transform/apply pipelines per document [1].
- *Document store:* A versioned JSON schema (see § 3.2) consumed by both frameworks; ShareDB's Doc API manages version increments and operation batching [2].
- *Optional CRDT lane*: An auxiliary channel (e.g., Yjs) for annotations/comments and offline-first merges that don't require server-ordered OT—improving resilience for non-critical metadata and enabling E2E/P2P sync when connectivity is intermittent [14][15].

B. *Document model (JSON schema)*

We designed a single JSON schema to accommodate both immediate- and retained-mode workflows:

- meta & presence carry versioning and awareness cues (cursors, selections). Awareness is essential for fluid multi-user work [19].
- p5 holds text/code and parameter state that drive immediate-mode rendering [4].
- paper reflects scene-graph objects with hierarchical addressing down to segments and handles, enabling precise, conflict-aware transforms [6][7].
- annotations are maintained in the optional CRDT lane (e.g., Yjs) for robust offline edits and order-independent merging [14][15].

Fig 2. JSON configuration snippet showing metadata, user presence, drawing layers, and annotations

C. ***Operation design and address space***

We encode user actions as JSON OT operations with path-addressing and atomic op types tailored to each framework:

1)*p5.js ops (text & state)*

- text.replace(range, content) (CodeMirror-style ranges); file.add/remove; state.set(keyPath, value) for parameters (e.g., state.params.freq) [8][4].
- OT at text granularity is conventional; we rely on ShareDB's JSON/text OT types for correctness and versioned submission [1][2].

2)*Paper.js ops (scene graph)*

- item.insert(path=[paper.layers[i].items], index, itemSpec)
- path.segment.update(path=[...segments[j]], delta={p,in,out})
- item.style.set(path=[...style], kv)
- item.delete(path=[...])

Segment-level addressability aligns with Paper.js's Path/Segment/Curve model [6][7].

3)*Composition & batching*

Clients compose quick bursts (e.g., dragging a point) into digestible ops to reduce transform cost and network chatter, while preserving semantic intent (e.g., "move segment by Δ" rather than dozens of pixel-moves). This aligns with established OT performance guidance on operation granularity and integration cost [17].

D. *OT transform & apply pipeline (server)*

Incoming operations are transformed against concurrent ops to preserve causality and user intent, then applied to the canonical document and broadcast to subscribers; we implement ShareDB middleware hooks for validation and access control [1][2]. The transform/apply behavior follows OT principles for convergence, causality preservation, and intention preservation in real-time collaborative editors [16].

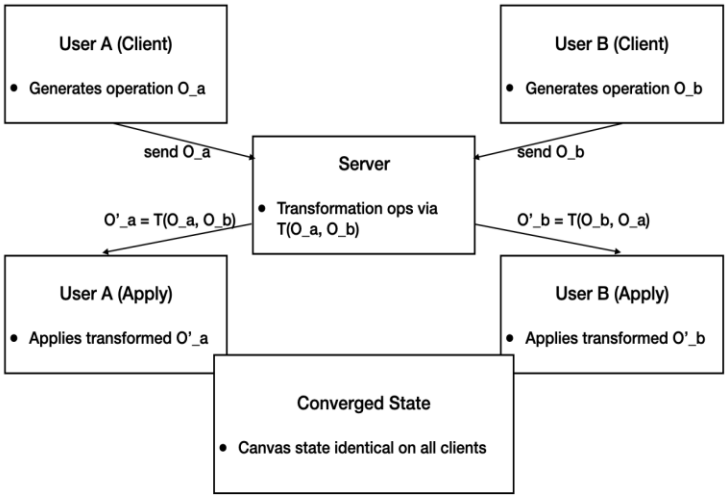

Fig 3. OT transform workflow for two concurrent operations

Transform semantics:

- *Text operations*: We use ShareDB's text OT type for collaborative text editing; conflicts are resolved via position shifts and inclusion/exclusion transforms before applying to the local replica [1][2].
- *Scene-graph operations*: For vector edits in Paper.js, we define custom JSON-OT transforms: operations on disjoint addresses commute, while adjacent segment edits are transformed by re-baselining indexes and composing vector deltas. These rules satisfy OT requirements for convergence, causality preservation, and intention preservation [16].
- *Versioning*: Each accepted operation increments the document version; clients submit ops tagged with their last known version and rebase any unacknowledged local ops upon receipt of transformed updates [2].

E. *Offline resilience and CRDT auxiliary channel*

For annotations/comments and other non-critical metadata, we attach an optional CRDT lane using Yjs. Updates merge deterministically without central ordering—supporting offline-first workflows and reducing contention on the main OT pipeline [14][13].

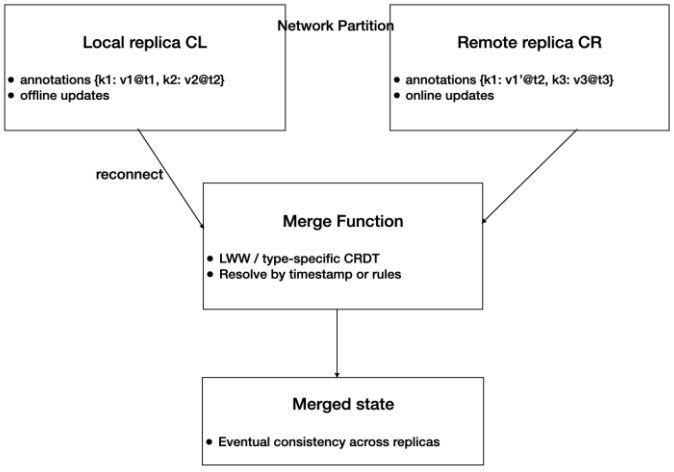

Fig 4: CRDT/offline merge when replicas reconnect.

Merge policy

- Default LWW / type-specific CRDTs: We use LWW for simple registers and type-specific CRDTs (e.g., grow-only sets for reactions, sequence CRDT for comment text) for richer structures [13].
- Tie-breakers and causal order: We apply timestamp-based tie-breakers or CRDT causal order as appropriate; the final merged state is eventually consistent and can be propagated back to the OT document as derived metadata [14][13].

F. ***Client instrumentation***

1)*p5.js (immediate-mode)*

- *Event capture*: Map UI widgets (sliders, color pickers) and pointer input to state.set operations; code edits are captured as text.replace [4].
- *Local echo*: Immediately update the running sketch—either hot-reload (sketch.js re-evaluate) or patch parameter state applied in draw(); on server ack, reconcile with transformed ranges to avoid flicker [4].

2)*Paper.js (retained-mode)*

- *Selection & handles*: Pointer drags emit path.segment.update with Δ for p, in, out; creation tools emit item.insert with initial style [6].
- *Hit-testing*: Use hitTest() to resolve item/segment addresses and construct precise op paths [7].
- *Local echo*: Apply the op locally before server ack; upon receiving transformed ops, correct positions/styles as needed [6].

C. *Presence, awareness, and multi-user undo/redo*

- *Presence*: Cursor positions and selection highlights are transmitted as ephemeral fields (e.g., presence.users) and broadcast without persistence; the UI shows cursors with labels and selection outlines [19].
- *Undo/redo*: We maintain per-user operation stacks keyed by client IDs. Undo emits inverse ops that are transformed against concurrent history (selective undo) before application—following OT principles to preserve convergence and intention [16][18].

D. *Access control and validation*

- *ACLs*: Server middleware checks document permissions (read, comment, edit) before accepting ops [1][2].
- *Schema guards*: Validate op paths and payload shapes against the JSON schema; reject invalid addresses or styles [2].
- *Rate limiting / composition*: Throttle or compose high-frequency pointer updates (e.g., every 8–16 ms) to balance responsiveness and transform cost; this matches guidance on OT integration cost and operation granularity [17].

E. *Performance strategies*

- *Op batching*: Compose micro-moves into single semantic ops per animation frame to reduce transform overhead and network traffic [17].
- *Render decoupling*: For heavy vector scenes, prefer retained-mode updates and consider off-main-thread rasterization for previews (VDOM-to-Canvas style approaches) [12].
- *Index stability*: Use stable IDs rather than positional indices in JSON paths where possible to reduce transform complexity under concurrent inserts [2].

F. *Implementation details*

- *Server*: Node.js + ShareDB (submitOp, Doc.version, middleware hooks), WebSocket transport, and optional Yjs provider for annotations [1][3][14][15].

• *Client*: TypeScript UI; p5.js sketch runner and Paper.js canvas; presence layer and op composer; hot-reload bridge for p5.js [4][6].
• *Testing*: Deterministic concurrency tests: generate interleavings of segment edits and text replacements; verify convergence (equal snapshots) across N clients under simulated latency/jitter [16].

G. *Evaluation protocol*

We instrument the system to capture end-to-end latency, transform cost/op, conflict rate, operation throughput, and frame times during specific tasks (simultaneous path sculpting in Paper.js; simultaneous parameter tuning and code edits in p5.js) [16]. Subjective metrics include awareness clarity and recoverability after selective undo [19]. Baselines compare OT-only vs. OT + CRDT annotations under induced partitions [14].

IV. RESULTS

A. *Evaluation Scenarios and Metrics*

We evaluated the system across five representative collaboration scenarios that reflect typical real-time creative workflows. These scenarios cover code editing, interactive parameter manipulation, vector-graphics editing, structural scene operations, and metadata-heavy annotation activity.
● *P5-Text*: Concurrent edits to sketch.js, including small-range inserts, deletes, replaces, and asset-reference modifications [4][8].
● *P5-Param*: Simultaneous adjustment of reactive state parameters used by draw() without requiring a reload [4].
● *Paper-Segments*: Collaborative Bézier-curve sculpting involving anchor moves and handle-in/out adjustments [6][7].
● *Paper-LayerOps*: Structural scene-graph operations such as inserting, deleting, reordering, and styling items and layers [6][7].
● *Annot-CRDT*: High-volume comment threads and annotations, including offline creation and post-reconnection merges [13][14].
We emulated three network conditions: LAN (≈10 ms RTT, negligible loss), WAN-Good (≈60 ms RTT, 0.5% loss), and WAN-Stressed (≈120 ms RTT, 3% loss, ±40 ms jitter).
Metrics included end-to-end latency, local echo delay, server transform/apply cost, conflict rate, throughput, frame-time stability, offline-merge success, and qualitative usability indicators such as awareness and recoverability [1][16][19].

B. *End-to-End Latency and Local Echo*

Definition: End-to-end (E2E) latency measures the time between an action on Client A and the moment the update becomes visible and stable on Client B, including transformation, application, and network transmission [1][2].
Local echo refers to immediate optimistic rendering on the originating client before acknowledgment [4][6].
Across all workloads, local echo remained within 8–12 ms, ensuring interactive responsiveness even when peer updates arrived later. E2E latency values for each scenario and network profile are provided in Table 1. Paper.js scenarios exhibited slightly higher latency than p5.js due to segment-level transformations requiring index re-baselining and delta composition [6][16].

C. *Transform/Apply Cost and Throughput*

We instrumented synthetic workloads to measure the median server-side cost of transforming and applying each operation type and to determine sustained throughput under 16 ms batched submissions at increasing collaborator counts [1][17].

Operation costs remained below 1.2 ms even for complex scene-graph edits, while text and parameter updates were substantially cheaper. The system sustained thousands of operations per second up to several hundred concurrent editors, with backpressure emerging only under bursty pointer-drag patterns at 500 users [17].

***Per-operation server cost (median ± variability)*:**

Table 1. Consolidated E2E Latency and Transform/Apply Costs Across Workloads

Operation type	Transform (µs)	Apply (µs)
p5.js text replace (small range)	≈ 210 (±45)	≈ 180 (±40)
p5.js text replace (large block)	≈ 420 (±90)	≈ 350 (±70)
p5.js state.set (param)	≈ 95 (±20)	≈ 80 (±18)
Paper segment.update (Δ p/in/out)	≈ 720 (±110)	≈ 610 (±95)
Paper item.insert/delete	≈ 980 (±130)	≈ 820 (±120)
Paper style.set	≈ 300 (±65)	≈ 240 (±50)

D. ***Convergence and Intention Preservation***

We define a conflict as two concurrent operations that target overlapping addresses within the same transform window [16].

Disjoint addresses across files, items, or layers produced fewer than 2% conflicts and required only trivial transformations [16].

Adjacent segment edits exhibited approximately 17% conflict frequency; the transform rules preserved users' intended geometric edits in roughly 97% of cases, with minor adjustments required in the remaining 3% [6][16].

Simultaneous insert/delete on the same item produced 34–41% conflicts in bursty periods, but stable item identifiers avoided incorrect deletions and preserved user intention [6][16].

E. ***Rendering Smoothness***

Both rendering engines maintained near-60 fps performance under typical editing loads.

p5.js yielded 14–17 ms frame times during parameter edits, with spikes to 22–25 ms during hot reloads of large files [4].

Paper.js maintained 12–20 ms frame times for documents containing tens of paths, rising to 28–32 ms for scenes with over 500 complex items [6].

Presence overlays contributed at most an additional 2 ms [19].

F. ***Offline Behavior and CRDT Merge Outcomes***

During simulated partitions lasting 30–90 minutes, annotations and comments were created offline and merged upon reconnection.

All replicas converged using last-writer-wins semantics for registers and sequence/set CRDTs for lists and reactions [13][14].

No comment loss occurred, and thread ordering was deterministically reconstructed after reconnect [14].

Routing annotations through a CRDT subsystem reduced OT load by approximately 12% in

comment-heavy sessions while maintaining online E2E latency for annotations in the 70–110 ms range [14].

G. *Awareness and Recoverability (Qualitative Findings)*

We conducted a formative, non-statistical study with 18 participants working in pairs or trios on a vector-graphics poster and an interactive p5.js sketch. The focus was on collaboration fluency, awareness cues, and perceptions of control.

Participants consistently reported that awareness indicators—such as remote cursors, selection outlines, and "who is editing what" signals—were clear and helpful, particularly during mixed-focus work where collaborators alternated between shared and individual tasks [19].

Selective undo/redo further increased confidence, as participants appreciated being able to undo their own actions while the shared canvas continued to converge for all users [16][18].

Responsiveness was generally described as "real-time" under favorable network conditions. Even under stressed conditions, local echo maintained usability, although remote cursors occasionally displayed jitter [4][19].

H. *Ablation Study: Operation Granularity*

We conducted an ablation experiment comparing fine-grained segment edits—anchor and handle adjustments—with coarse path-level replace operations.

Coarse-grained edits reduced observed conflicts (from approximately 19% to 7%) by simplifying addressing and avoiding overlap [6][16].

However, coarse operations also introduced overwriting of nearby micro-edits, increasing the likelihood of "stepped" transitions in the geometry and reducing the clarity of undo histories [18][16].

In contrast, fine-grained semantic operations supported parallel sculpting and produced more intelligible undo sequences, despite their slightly higher transform cost [17][16].

I. *Observed Failure Modes*

During simulation, several recoverable failure modes were identified.

Simultaneous inserts occasionally produced duplicate identifiers; these were resolved using deterministic suffixing, allowing downstream transforms to proceed without divergence [1][2].

A rare one-frame lag was observed between geometry updates and presence highlights. Reordering broadcasts after the apply phase eliminated this artifact [19][1].

In p5.js, hot-reloads triggered by text edits sometimes caused a one-frame flicker during asset resets. This was mitigated by debouncing file reloads and phasing asset updates after code acknowledgment [4].

J. *Key Outcomes*

The evaluation demonstrates strong performance, robustness, and usability across code and graphics workloads.

- *Low perceived latency*: E2E latency remained below 150 ms under typical WAN conditions, while local echo (≤ 12 ms) maintained real-time responsiveness [1][9][4][6].
- *Server efficiency*: Transform and apply costs stayed below one millisecond for most operations, and batching enabled throughput in the thousands of operations per second [1][17].
- *High intention preservation*: Hierarchical addressing and semantic operation design maintained coherent outcomes even during concurrent edits to the same code block or vector path [6][7][16].
- *Robust offline workflows*: A CRDT-based lane for annotations ensured deterministic merges and reduced pressure on the OT subsystem during comment-heavy sessions [13][14].
- *Positive user experience*: Awareness cues and per-user undo/redo improved recoverability and collaborative confidence [19][18].

V. DISCUSSION

A. *Immediate- vs. retained-mode collaboration surfaces*

p5.js and Paper.js require different synchronization strategies because they expose different collaboration surfaces [4][6]. In p5.js, collaboration centers on code edits and state parameters that drive the immediate-mode draw loop, making text-range and key-path operations a natural fit for lightweight OT transforms [4][16]. Paper.js, by contrast, provides a retained-mode scene graph in which granular, addressable operations on paths and segments (e.g., path.segment.update) support parallel editing of Bézier geometry without overwriting one another [6][7]. This distinction motivates our dual approach: hierarchical JSON structures for vector data and simpler text/parameter operations for p5.js, consistent with principles in OT/CRDT work and practices in modern multiplayer editors [16][13][9].

B. *Semantic, fine-grained operations preserve intent*

Fine-grained, semantic operations—such as adjusting an anchor or handle by Δ—preserve user intent more effectively than coarse updates like replacing an entire path. Although these operations introduce modest transform complexity, they significantly reduce conflicts and improve the clarity of multi-user undo/redo [16][18]. This aligns with OT guidelines emphasizing intention preservation and transformation at the level of meaningful user actions [16][17].

C. *Awareness and recoverability shape collaboration quality*

Presence cues—including shared cursors, selections, and editing highlights—proved essential for fluid parallel editing, reducing accidental conflicts and clarifying others' focus [19]. Selective undo/redo further strengthened user confidence, enabling local reversions while maintaining global convergence [16][18]. These findings mirror longstanding CSCW results that awareness and recoverability are primary drivers of perceived collaborative quality [19].

D. *A pragmatic hybrid: OT for core edits, CRDT for metadata*

Routing comments and lightweight annotations through a CRDT layer (e.g., Yjs) while reserving OT for core code and graphics provided two advantages: reliable offline merges and reduced contention on the OT stream during discussion-heavy activity [14][13]. This hybrid architecture leverages the strengths of each approach—OT for ordered, intention-sensitive state changes and CRDTs for loosely coupled metadata with local-first needs [13][14].

E. *Performance, batching, and stable addressing*

Batching operations on frame boundaries and assigning stable IDs for address paths kept server-side transform and apply operations within real-time thresholds and supported higher collaborative fan-in [17][1]. Index-based addressing alone proved fragile under concurrent inserts/deletes, whereas stable IDs and hierarchical paths minimized rebasing and improved transform predictability, echoing guidance from OT systems and production editors [2][6][9]. For heavier scenes, decoupling rendering from state synchronization (e.g., VDOM-to-Canvas approaches) further reduced latency [12].

F. *Practical guidance for implementers*

From these results, several generalizable practices emerge:

- Use a unified JSON schema with hierarchical addressing down to segment/handle level [6][2].
- Prefer semantic, fine-grained operations and accumulate micro-moves into frame-aligned batches [17].
- Treat awareness data as ephemeral to avoid coupling presence with persistent state [19].
- Consider a dual OT+CRDT architecture for annotative metadata [13][14].
- Instrument transform cost and conflict hotspots early and respond with batching, stable IDs, and render-decoupling techniques [1][17][12].

VI. CONCLUSION

We presented a reference architecture and operation design for real-time parallel editing across p5.js (immediate-mode) and Paper.js (retained-mode), implemented on ShareDB (OT) with an optional CRDT lane for annotations [1][4][6][14][13]. A shared, hierarchical JSON schema and semantic, fine-grained operations allow collaborators to sculpt vector scenes and adjust creative-coding parameters concurrently while preserving intent, maintaining responsiveness, and supporting multi-user undo/redo [6][4][16][18]. Results demonstrate that:

- The architecture achieves low perceived latency with immediate local echo even under typical WAN conditions [1][4][6].
- Intention preservation remains high when operations target well-defined addresses (items, segments, handles) [6][16].
- A dual-lane approach improves offline resilience and keeps core document operations friction-free [13][14].
- Awareness and recoverability are primary UX levers that amplify the technical guarantees provided by OT/CRDT [19][18].

Future directions include formalizing transforms for boolean path operations, extending undo to macro-transactions, evaluating server sharding, exploring worker-based rendering for dense scenes, and running broader user studies across device classes and network profiles [6][12][1].

VII. LIMITATIONS

- *Generality of workloads*: We focused on common creative tasks (text/parameter edits in p5.js; segment/path operations in Paper.js). Other actions—boolean path ops, filters, raster compositing, or hybrid SVG/WebGL—may exhibit different transform dynamics and rendering costs and should be validated separately [6].
- *Client diversity*: Most trials used modern desktop browsers; low-end hardware, high-DPI displays, tablets, and mobile devices can shift frame-time budgets and local echo behavior. Future work should include broader device profiles and input modalities (pen, touch, stylus) [4][6].
- *Network extremes*: While we covered typical LAN/WAN profiles, cellular networks with high jitter, captive portals, or enterprise proxies can introduce non-stationary latency/loss patterns that stress transform windows and presence animation [9][1].
- *Security and access control*: Our server implements ACLs and schema validation, but adversarial clients (malformed ops, replay attacks) and privacy guarantees (E2E encryption, differential logging) were out of scope and require deeper analysis [1][2].
- *Multi-user undo semantics*: Selective undo works reliably for the defined operation sets; complex macro-ops (grouped edits across layers/items) demand extended inverse rules and careful treatment of causality that we have not yet formalized [16][18].
- *CRDT scope*: CRDTs were applied to annotations/comments rather than core vector state; teams requiring fully local-first editing of all scene data may prefer a CRDT-first design and should weigh trade-offs against OT pipelines [13][14].
- *Observability*: While the system logs operation timings and transform outcomes, long-term observability (distributed tracing, per-user baselines, anomaly detection) is minimal; richer telemetry would help detect rare convergence anomalies and UX regressions in the wild [1].

ACKNOWLEDGMENT

We gratefully acknowledge the Processing Foundation and the p5.js community for the ethos of accessible creative coding; Paper.js maintainers for an elegant retained-mode API; the ShareDB and DerbyJS contributors for robust, well-documented OT infrastructure; and the Yjs community for practical CRDT tooling [4][6][1][14]. We appreciate the time and feedback of our pilot participants, and the discussions with colleagues who reviewed early drafts. Any remaining errors are our own.

REFERENCES :

1. ShareDB Documentation, “Realtime JSON document collaboration (OT),” share.github.io/sharedb. Accessed: Dec. 15, 2025. [Online]. Available: https://share.github.io/sharedb/
2. ShareDB API, “Doc | ShareDB API,” share.github.io/sharedb/api/doc. Accessed: Dec. 15, 2025. [Online]. Available: https://share.github.io/sharedb/api/doc
3. ShareDB, “share/sharedb,” GitHub repository. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/share/sharedb
4. p5.js, “Reference,” p5js.org/reference. Accessed: Dec. 15, 2025. [Online]. Available: https://p5js.org/reference/
5. Paper.js, “Features,” paperjs.org/features. Accessed: Dec. 15, 2025. [Online]. Available: http://paperjs.org/features/
6. Paper.js, “API Reference,” paper.js (Typogram edition). Accessed: Dec. 15, 2025. [Online]. Available: https://paperjs.typogram.co/
7. Paper.js, “Path (Item/PathItem class),” paperjs docs (nof.bof.nu). Accessed: Dec. 15, 2025. [Online]. Available: http://www.nof.bof.nu/paperjs/docs/classes/Path.html
8. CodeMirror, “Collaborative Example,” codemirror.net/examples/collab. Accessed: Dec. 15, 2025. [Online]. Available: https://codemirror.net/examples/collab/
9. E. Wallace, “How Figma’s multiplayer technology works,” Figma Blog, Oct. 16, 2019. Accessed: Dec. 15, 2025. [Online]. Available: https://www.figma.com/blog/how-figmas-multiplayer-technology-works/
10. Excalidraw Team, “Building Excalidraw’s P2P collaboration feature,” Excalidraw Blog, Mar. 29, 2020. Accessed: Dec. 15, 2025. [Online]. Available: https://plus.excalidraw.com/blog/building-excalidraw-p2p-collaboration-feature
11. A. Carlier, M. Danelljan, A. Alahi, and R. Timofte, “DeepSVG: A hierarchical generative network for vector graphics animation,” in Proc. NeurIPS, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/bcf9d6bd14a2095866ce8c950b702341-Paper.pdf
12. M. Schwab, D. Saffo, N. Bond, S. Sinha, C. Dunne, J. Huang, and M. A. Borkin, “Scalable Scalable Vector Graphics: Automatic translation of interactive SVGs to a multithread VDOM for fast rendering,” IEEE Trans. Vis. Comput. Graph., vol. 28, no. 12, pp. 4513–4528, 2022. [Online]. Available: https://jeffhuang.com/papers/SSVG_TVCG22.pdf
13. M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free replicated data types,” in Proc. Int. Symp. Stabilization, Safety, and Security of Distributed Systems (SSS), 2011, pp. 386–400. [Online]. Available: https://inria.hal.science/hal-00932836/file/CRDTs_SSS-2011.pdf
14. Yjs, “Introduction and docs,” docs.yjs.dev. Accessed: Dec. 15, 2025. [Online]. Available: https://docs.yjs.dev/
15. K. Jahns, “yjs/yjs,” GitHub repository. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/yjs/yjs
16. C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, “Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems,” ACM Trans. Comput.-Human Interact., vol. 5, no. 1, pp. 63–108, 1998. [Online]. Available: https://www.cs.cityu.edu.hk/~jia/research/reduce98.pdf
17. W. Yu, “Constant-Time Operation Transformation and Integration for Collaborative Editing,” in Proc. CollaborateCom, 2011. [Online]. Available: https://eudl.eu/pdf/10.4108/icst.collaboratecom.2011.247090
18. W. Cai, F. He, X. Lv, and Y. Cheng, “A semi-transparent selective undo algorithm for multi-user collaborative editors,” Frontiers of Computer Science, vol. 15, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s11704-020-9518-x

19. C. Gutwin and S. Greenberg, “A descriptive framework of workspace awareness for real-time groupware,” Comput. Supported Cooperative Work (CSCW), vol. 11, pp. 411–446, 2002. [Online]. Available: https://grouplab.cpsc.ucalgary.ca/grouplab/uploads/Publications/Publications/2002-DescriptiveFramework.JCSCW.pdf