```
import { useCallback, useEffect, useRef, useState } from 'react';
import isEqual from 'lodash/isEqual';
import debounce from 'lodash/debounce';
import { Node, Edge } from 'reactflow';

interface AutosaveHookOptions {
  nodes: Node[];
  edges: Edge[];
  currentFlowId: string | null;
  taskResults: Record<string, string>;
  onSave: (data: any) => Promise<any>;
  debounceMs?: number;
  maxRetries?: number;
  enabled?: boolean;
}

interface SaveQueueItem {
  nodes: Node[];
  edges: Edge[];
  taskResults: Record<string, string>;
  timestamp: number;
}

export const useEnhancedAutosave = ({
  nodes,
  edges,
```

```
  currentFlowId,
  taskResults,
  onSave,
  debounceMs = 1000,
  maxRetries = 3,
  enabled = true
}: AutosaveHookOptions) => {
  // Refs for tracking state and save status
  const saveInProgressRef = useRef(false);
  const saveQueueRef = useRef<SaveQueueItem[]>([]);
  const previousStateRef = useRef({
    nodes: [] as Node[],
    edges: [] as Edge[],
    taskResults: {} as Record<string, string>
  });
  const retryCountRef = useRef(0);
  const [lastSaveStatus, setLastSaveStatus] = useState<'success' | 'error' | null>(null);

  // Deep comparison utility
  const hasStateChanged = useCallback((current: any, previous: any) => {
    return !isEqual(current, previous);
  }, []);

  // Process save queue
  const processSaveQueue = useCallback(async () => {
    if (!enabled || saveInProgressRef.current || saveQueueRef.current.length === 0) {
```

```
  return;
}

try {
  saveInProgressRef.current = true;
  const latestState = saveQueueRef.current[saveQueueRef.current.length - 1];

  // Attempt to save
  await onSave({
    flow_id: currentFlowId,
    nodes: latestState.nodes,
    edges: latestState.edges,
    results: latestState.taskResults
  });

  // Success handling
  retryCountRef.current = 0;
  previousStateRef.current = {
    nodes: JSON.parse(JSON.stringify(latestState.nodes)),
    edges: JSON.parse(JSON.stringify(latestState.edges)),
    taskResults: JSON.parse(JSON.stringify(latestState.taskResults))
  };
  saveQueueRef.current = [];
  setLastSaveStatus('success');

} catch (error) {
```

```
    // Error handling with retry logic
    if (retryCountRef.current < maxRetries) {
      retryCountRef.current++;
      setTimeout(() => {
        saveInProgressRef.current = false;
        processSaveQueue();
      }, Math.pow(2, retryCountRef.current) * 1000); // Exponential backoff
    } else {
      setLastSaveStatus('error');
      console.error('Max retry attempts reached:', error);
      saveQueueRef.current = [];
    }
  } finally {
    if (retryCountRef.current === 0) {
      saveInProgressRef.current = false;
    }
  }
}, [currentFlowId, enabled, maxRetries, onSave]);

// Optimized debounced save function
const debouncedSave = useCallback(
  debounce(() => {
    if (!enabled || !currentFlowId) return;

    const currentState = {
      nodes,
```

```
      edges,
      taskResults
    };

    // Check if state has actually changed
    if (!hasStateChanged(currentState, previousStateRef.current)) {
      return;
    }

    // Add to save queue
    saveQueueRef.current.push({
      ...currentState,
      timestamp: Date.now()
    });

    // Trim queue to keep only latest states if queue gets too long
    if (saveQueueRef.current.length > 10) {
      saveQueueRef.current = saveQueueRef.current.slice(-10);
    }

    processSaveQueue();
  }, debounceMs),
      [currentFlowId, nodes, edges, taskResults, enabled, debounceMs, hasStateChanged,
processSaveQueue]
  );
```

```
  // Setup effect for autosave
  useEffect(() => {
    if (!enabled || !currentFlowId) return;

    debouncedSave();

    return () => {
      debouncedSave.cancel();
      // Attempt to save any pending changes on unmount
      if (saveQueueRef.current.length > 0) {
        processSaveQueue();
      }
    };
  }, [enabled, currentFlowId, nodes, edges, taskResults, debouncedSave, processSaveQueue]);

  // Return save status and control functions
  return {
    lastSaveStatus,
    forceSave: () => {
      debouncedSave.cancel();
      processSaveQueue();
    },
    isSaving: saveInProgressRef.current
  };
};
```