

Permalink URI:

vscode://joelabenhaim.aicode-ja/chat?sessionId=2d6b233d88f04687b47cd547ba464664-VS-V%C3%A9rifier-finalise

<USER>

[Verify]SPECIF FINALISER LIVESESSION POUR LES SESSIONS PROMPT NORMALES 20260712 204731

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>

🔧 ☒ scanning recent files

🔧 ☒ scanning git: 📁 "État Git courant", 📁 "Historique récent", 📁 "Portée des changements"

🔧 ☒ reading project: 📄 [liveSession.programming-doc-readme.md](#), 📄 [easyAgents.programming-doc-readme.md](#), 📄 [isomorphism-invariant-rule-between-streaming-and-rehydration-of-chat-sessions.programming-doc-readme.md](#), 📄 [thinking-bubble-and-hourglass-streaming-rules-invariants.programming-doc-readme.md](#), 📄 [prompt-stream-dispatch-lock.programming-doc-readme.md](#), 📄 [fallback-persist-and-title.programming-doc-readme.md](#)

🔧 ☒ scanning git: 📁 "Inventaire du commit", 📁 "Résumé quantitatif", 📁 "Statistiques par fichier", 📁 "Différence avec la base"

🔧 ☒ reading

project:

📄 [LiveSessionRun.type.ts](#),
📄 [LiveSessionPanel.type.ts](#),
📄 [LiveSessionOpenResult.type.ts](#),
📄 [LiveSessionReplay.type.ts](#),
📄 [SidebarStore.ts](#),
📄 [SidebarUiMessages.ts](#), 📄 [parseSlimbkStreamBodyError.util.ts](#), 📄 [aiStopSuffix.util.ts](#), 📄 [LiveSessionRecord.type.ts](#),
🔗 e "SourceId dans le bus", 🔗 e "Contexte origine stream", 🔗 e "Descripteur live strict", 🔗 e "Start accepté synchrone", 🔗 e "Tentative de replay", 🔗 h "Identité tentative Open", 🔗 e "Nom streamKey résiduel", 🔗 e "Identifiant stream instance", 🔗 e "Identifiant source exact",

📄 [usePromptLiveSession.hook.ts](#),
📄 [PromptLiveSession.types.ts](#),
📄 [NormalPromptLiveSession.adapter.ts](#),
📄 [PromptLiveSessionBootstrap.adapter.ts](#),
📄 [EasyAgentsBatchLiveSession.adapter.ts](#),
📄 [AiStreamStartArgs.type.ts](#),
📄 [applyHydratedPromptSessionState.util.ts](#),
📄 [usePromptRunCommands.hook.ts](#),
📄 [preparePromptStreamingTurn.util.ts](#),
📄 [preparePromptContinuationStreaming.util.ts](#),
📄 [usePromptThinkingLifecycle.hook.ts](#), 📄 [usePromptRunController.finalization.hook.ts](#), 🔗 e "Type PromptMessage meta", 🔗 e "Casts inconnus ajoutés", 🔗 e "Annotations any", 🔗 e "Désactivation ESLint", 🔗 e "Import dynamique"

🔧 ☒ scanning git: 📁 "Historique du cast"

🔧 ☒ reading project: 📄 [useAiStream.hook.ts](#), 🔗 e "Réservation de source", 🔗 e "Promotion du run", 🔗 e "Finalisation forwarding", 🔗 e "Claim destructif", 🔗 e "Éviction par claim", 🔗 e "Claim abort source", 🔗 e "Claim abort viewer", 🔗 e "Permit ouverture", 🔗 e "Publication baseline", 🔗 e "Append buffer", 🔗 h "API registre ouverture", 🖱️ "État interne du registre", 📄 [LiveSessionRegistry.service.ts](#) [L720-L920],

📄 [NormalPromptLiveSessionSource.service.ts](#),
📄 [LiveSessionSourceLifecycle.service.ts](#),
📄 [stream.endpoint.ts](#),
📄 [iaStreamHandlers.ts](#),
📄 [liveSessionStreamHandlers.ts](#),
📄 [LiveSessionOpen.service.ts](#),
📄 [PromptLiveSessionPanelQuery.util.ts](#),
📄 [PromptLiveSessionPanelCoordinator.service.ts](#),
📄 [PromptLiveSessionDetachedStop.service.ts](#),

[openPanel.util.ts](#),
[PromptThreadCleanup.manager.ts](#),
[SidebarQueryRunningBridge.ts](#),
[SidebarActivityQuery.tsx](#), [SidebarStore.queryWindow.mutations.manager.ts](#), [SidebarStore.handlers.manager.ts](#),
[SidebarStore.core.manager.ts](#) [L1-L160], [SidebarStore.core.manager.ts](#) [L160-
L220], [SidebarStore.queryWindow.listeners.manager.ts](#), ["Réhydratation activités"](#), ["Snapshots live actifs"](#), ["Observer fermé interdit"](#), ["Terminal non finalisé"](#), ["Stop demandé réhydraté"](#), ["Projection Activité réhyd"](#), [SidebarStore.rehydrate.manager.ts](#) [L300-
L520],
[delete.endpoint.ts](#),
[EasyAgentsBatchRunRetry.service.ts](#), [EasyAgentsBatchLiveSessionAdapter.service.ts](#), [abortSource.endpoint.ts](#),
[PromptLiveSessionDetachedStop.service.ts](#),
[Prompt.tsx](#),
[PromptInput.tsx](#),
[usePromptCancelController.hook.ts](#),
[usePromptRunController.hook.ts](#), [usePromptIndexPending.hook.tsx](#), ["Refus start dans Prompt"](#), ["Double soumission Prompt"](#), ["Masque transactionnel"](#), ["Barrière refusée"](#), [PromptFallbackPersistBarrier.testHarness.tsx](#) [L1-
L220], [PromptFallbackPersistBarrier.testHarness.tsx](#) [L440-L514], [usePromptSessionLocator.hook.ts](#) [L1-
L160], ["Snapshot brut figé"](#), [usePromptSessionLocator.hook.ts](#) [L275-L310], ["Soumission VscTextarea"](#), [VscTextarea.tsx](#) [L500-
L720],
[extTypedMessageBus.util.ts](#), [extMessageBus.util.ts](#), [MessageRegistry.ts](#), [SlimbkResult.type.ts](#), [Slimbk.ts](#),
[slimbkBuildHeader.ts](#), ["Échec démarrage son"](#), ["Erreur son auxiliaire"](#), ["Mock son qui lève"](#), [PromptQueryDoneSound.manager.ts](#), [OrchMessagesMemoryAssistantPatch.ts](#) [L1-
L200], [OrchMessagesMemoryAssistantPatch.ts](#) [L200-L360], [OrchMessagesMemoryAssistantPatch.ts](#) [L360-
L450], ["ID frontend côté extension"](#), ["ID Open côté backend"](#), ["Ancien streamKey"](#), ["Ancien windows Activity"](#), ["Ancien type Activity"](#), [LiveSessionService.ts](#), [LiveSessionReplay.service.ts](#), [ChatSessionLocator.type.ts](#) [L1-
L120], ["openCentralPanel"](#), [registerDeepLinkUriHandler.ts](#) [L120-
L165], [useAiStreamQueue.hook.ts](#), [usePromptStreamDispatchGateQueue.hook.ts](#), [PromptStreamLockStore.ts](#)
🔧 ☒ scanning git: 🌀 ["Évolution de Queue B"](#)
🔧 ☒ reading project: ["Finalisation transport vide"](#), ["Stream vide finalisé"](#), ["Finalized sans chunks"](#), [EasyAgentsBatchRunChildExecution.service.ts](#) [L1-L260], ["Succès inscription source"](#), [EasyAgentsBatchRunExecution.service.ts](#) [L290-L390], [EasyAgentsBatchRunExecution.service.ts](#) [L390-
L440], ["Réouverture après Stop"](#), ["StopRequested dans Open"](#), ["AbortRequested dans Open"](#), ["Reopened dans tests Stop"](#), ["Fermeture après Stop"](#), [PromptLiveSessionDetachedStop.service.test.ts](#) [L90-L170]
🔧 ☒ scanning git: 🌀 ["Style du bandeau Prompt"](#)
🔧 ☒ reading project: ["Option disableAfterClick"](#), [PromptLiveSessionBanner.tsx](#), [usePromptRunController.uiState.hook.ts](#) [L1-
L200], [useStreamListener.hook.ts](#), ["Réception stream frontend"](#), [frontMessageBus.util.ts](#) [L1-
L260], [frontMessageBus.util.ts](#) [L260-L430], ["Chunk mauvais turnId"](#)
🔧 ☒ scanning git: 🌀 ["Anomalies de diff"](#), 🌀 ["Fichiers générés modifiés"](#), 🌀 ["Tests ajoutés et modifiés"](#)
🔧 ☒ reading project: ["Tests Stop normalPrompt"](#), [usePromptLiveSession.normalPrompt.test.tsx](#) [L150-L250]
🔧 ☒ scanning git: 🌀 ["Ajustement de l'index-gât"](#)
🔧 ☒ reading project: ["Leases vidées à éviction"](#), ["Replay après éviction"](#), ["Éviction test lease active"](#), ["Replay éviction publique"](#), [LiveSessionReplay.service.test.ts](#) [L70-
L130], [LiveSessionReplay.service.test.ts](#) [L1-L75]

Base factuelle de la vérification

Joel, cette spécification ressemble à une centrale nucléaire asynchrone — mais une centrale où chaque électron possède heureusement son propre `sourceId` 😊. Le niveau de détail est impressionnant.

L'audit porte sur l'état propre du dépôt au commit **967468bf**, dont le nom correspond exactement à celui demandé par la spécification.

Élément vérifié	Constat
État Git	✓ Arbre de travail propre, HEAD synchronisé avec la branche distante
Taille de l'implémentation	189 fichiers modifiés , environ 22 945 insertions et 11 850 suppressions
Couverture des fichiers planifiés	✓ Je n'ai identifié aucun groupe de fichiers prévu complètement absent
Tests ajoutés ou adaptés	✓ Près de 90 fichiers de test concernés
Qualité mécanique du diff	✓ Aucun défaut remonté par la vérification Git du diff
Portée supplémentaire	⚠ Le commit étend aussi le framework Slimbk, son générateur client, Queue A/Queue B et plusieurs harness de tests, ce qui est globalement justifié par les invariants demandés

J'ai également lu les documents d'architecture concernés, notamment :

- [documentation générique liveSession](#)
- [documentation EasyAgents](#)
- [invariant d'isomorphisme streaming/hydratation](#)
- [invariants de la bulle thinking](#)
- [documentation Queue B](#)
- [documentation fallback persist](#)

1) « L'ensemble de la spécification est-il implémenté ? »

Réponse

⚠ **Non, pas entièrement au sens contractuel strict.**

La quasi-totalité de l'architecture, des contrats, des services, des intégrations et des tests demandés est bien présente.

Aucun pan fonctionnel majeur n'est simplement absent.

Mon estimation est la suivante :

Mesure	Estimation
Couverture structurelle des modules et fichiers demandés	≈ 99 %
Couverture fonctionnelle des comportements demandés	≈ 96 %
Conformité comportementale stricte, en tenant compte des courses restantes	≈ 93–94 %
Complétude globale	≈ 96 %

La réponse reste **non** parce que plusieurs invariants explicitement demandés peuvent encore être violés :

1. une éviction destructive ne révoque pas réellement les leases déjà attachées ;
2. le bouton Stop peut redevenir actif après réouverture alors que l'abort source a déjà été revendiqué ;
3. une erreur synchrone dans `onData` publie trop tôt la fin frontend ;
4. la barrière du compositeur n'est pas totalement protégée contre une soumission refusée concurrente.

Conclusion de la question 1

La tranche est structurellement complète, mais elle ne peut pas être considérée comme implémentée à 100 % tant que ces courses ne sont pas corrigées.

2) « L'implémentation actuelle fait-elle ce qu'elle est censée faire selon la spécification ? Pour chaque fonctionnalité technique demandée, est-elle complète ? »

Évaluation détaillée par groupe fonctionnel

Fonctionnalité technique	Éléments vérifiés	État	C
Identité exacte <code>locator</code> + <code>turnId</code> + <code>sourceId</code>	contrat des runRefs	✓ Complète	10
Différenciation <code>turnId</code> logique / <code>sourceId</code> exécutionnel	Clés structurelles exactes et propagation multi-couche	✓ Complète	10
Union stricte des descripteurs EasyAgents/normalPrompt	contrat des panneaux liveSession	✓ Complète	10
Détection des doublons, paramètres inconnus et queries partielles	parse extension , dispatcher frontend	✓ Complète	10
Frontière React externe pour les queries invalides	intégration Prompt	✓ Complète	10
Désactivation de l'hydratation classique et de l'auto-run en mode live	intégration Prompt , hook d'hydratation	✓ Complète	10
Réservation synchrone, promotion, époque, tombstones et	registre liveSession , types	✓ Très	98

Fonctionnalité technique	Éléments vérifiés	État	C
retraites	internes du registre	largement complète	
Claims destructifs et génération destructive distincte	Même registre et façade liveSession	✓ Complète	10
Éviction destructive des records et leases	Registre + replay	✗ Lease attachée non réellement révoquée	82
Buffer ordonné, non compressé et non borné	Registre et service de replay	✓ Complète	10
Replay backlog puis tail, avec curseur par viewer	service de replay	△ Correct sauf après éviction	94
Replay <code>replaceTurnAssistants / continueAssistant</code>	service Open , contrat replay	✓ Complète	10
Lifecycle générique du <code>Readable lazy</code>	lifecycle source partagé	✓ Complète	10
Source Prompt normale et baseline exacte	service source normalPrompt	✓ Complète	99
Persistance de tout <code>meta.error</code> avant <code>append</code>	Même service et patch de la mémoire assistant	✓ Complète	10
Synthèse <code>SYSTEM_ERROR/INTERRUPTED</code> sans double terminal	Service source et contrats de suffixes	✓ Complète	10
Premier <code>meta.error</code> encodé dans le header privé V1	endpoint source Prompt , encodeur et décodeur V1	✓ Complète	10
<code>meta.error</code> tardif conservé dans le body après <code>200_OK</code>	Endpoint source et handlers extension	✓ Complète	10
Détection de l'enveloppe Slimbk non-200 post-header	parseur d'erreur body Slimbk	✓ Complète	10
Transport Slimbk permettant un header non-200 avec body maintenu	framework Slimbk , générateur du client Slimbk	✓ Complète	10
Origine exacte du stream par webview extension-only	bus typé extension , initialisation du bus extension	✓ Complète	10
Refus d'un message provenant d'un ancien panneau remplacé	handler source Prompt , handler de replay	✓ Complète	10
Allocation synchrone Activité avant publication auxiliaire	runtime Activité	✓ Complète	10
Union Activité Idle/observer/source/open/closed	contrat SidebarStore	✓ Complète	10
Idle si panneau ouvert, suppression si panneau fermé	transitions Activité	✓ Complète	10
Réhydratation Activité depuis panneaux, records et réservations	réhydratation Sidebar	✓ Complète	99
Ouverture Activité via RPC dédié	handlers Sidebar	✓ Complète	10
Stop détaché uniquement pour source fermée	UI Activité , service Stop détaché	✓ Présent	98
Stop source exact pour réservation ou record	endpoint abortSource	✓ Complète	10
Conservation du Stop désactivé après réouverture	Contrat Open + hook frontend	✗ Course restante	75
Coordinateur panneau par locator, ouverture/fermeture/settlement	coordinateur des panneaux Prompt	✓ Complète	98
Cleanup différé sous claim destructif	Coordinateur et cleanup de thread Prompt	✓ Complète	98
Façade publique async <code>openCentralPanel</code>	ouverture centrale des panneaux	✓ Complète	10
Démarrage frontend synchrone et single-flight	contrat de démarrage	✓ Complète	99

Fonctionnalité technique	Éléments vérifiés	État	C
Zéro scaffold sur refus pour Send/SPECIF/Retry/Edit/Reply/Continue	stream , hook stream commandes Prompt	✓ Complète dans les commandes	10
Guards <code>sourceId</code> + <code>streamInstanceId</code> dans Queue A et Queue B	Queue A , Queue B	✓ Complète	10
Publication de l'outcome après Queue A, Queue B et commit	hook stream	✓ Sur les chemins normaux	96
Erreur synchrone de <code>onData</code> transformée proprement en <code>transportError</code>	Hook stream + bus frontend	✗ Chemin incomplet	70
Handoff direct sans second Open	hook liveSession	✓ Complète	10
<code>preserveLocal</code> sans lecture du brouillon ni des attachments	application de snapshot hydraté	✓ Complète	10
Retry terminal sans stream fictif	Hook liveSession	✓ Complète	10
Composer éditable après armement, Send toujours verrouillé	adaptateur normalPrompt , PromptInput	✓ Complète	10
Stop unique dans le compositeur	Adaptateur normalPrompt et PromptInput	✓ Complète	10
Bannière Retry normalPrompt sans second Stop	bannière liveSession	✓ Complète	10
Barrière <code>fallback_persisted</code> corrélée à la tentative	intégration Prompt	✓ Présente	96
Démasquage sans effacement en l'absence d'ack	Même intégration Prompt	✓ Complète sur le chemin normal	10
Protection de cette barrière contre une seconde soumission refusée	Intégration Prompt	△ Course défensive restante	82
Continuation sur assistant existant et nouvelle bulle thinking	préparation de continuation	✓ Complète	10
Thinking réinitialisé par <code>streamInstanceId</code> et séparateur unique	lifecycle thinking	✓ Complète	10
Delete durable sous claim et attente du settlement source	endpoint Delete	✓ Implémenté	96
Retry EasyAgents sous claim exact	service Retry EasyAgents	✓ Implémenté	97
Préservation de l'UX et des règles EasyAgents	adaptateur EasyAgents backend , adaptateur EasyAgents frontend	✓ Complète	99
Tests d'isomorphisme stream/replay/hydratation	tests d'isomorphisme normalPrompt	✓ Très forte couverture	98
Documentation de maintenance	Documents d'architecture listés plus haut	✓ Complète et utile	98

Conclusion de la question 2

Toutes les grandes fonctionnalités ont été codées, mais quatre chemins de course ou d'erreur empêchent encore d'affirmer que chaque invariant fonctionne systématiquement comme spécifié.

3) « Y a-t-il des bugs ? »

Réponse

✓ Oui. J'ai identifié quatre anomalies factuelles, dont trois certaines et une course défensive étroite mais réelle.

Bug 1 — Une éviction ne révoque pas les leases d'un replay déjà attaché

Propriété

Évaluation

Gravité ● Élevée

Confiance 100 %

Zone Registre et replay

Contrat violé L'éviction destructive doit retirer records, leases et retraites replayables

Dans [le registre liveSession](#), `evictRecord(...)` :

- marque le record `evicted` ;
- le retire des index du registre ;
- retire les correspondances globales des lease IDs ;
- **mais ne vide pas `record.activeLeases`.**

Or :

- `readSubscription(...)` continue à considérer la lease comme active en consultant précisément `record.activeLeases` ;
- `isSubscriptionReplayAuthorized(...)` consulte la même map ;
- [le service de replay](#) vérifie l'éviction **après** avoir transmis les chunks du suffixe courant.

Scénario défectueux

1. Un viewer a consommé une partie du backlog.
2. D'autres chunks restent non lus.
3. Delete, Retry ou expiration de rétention évince le record.
4. Le waiter est réveillé.
5. La lease paraît encore autorisée.
6. Les chunks restants sont encore émis.
7. Le replay ne se termine qu'après ce drainage stale.

Le test actuel d'éviction ne couvre que le cas où aucun chunk non consommé ne reste au moment de l'éviction.

Impact

- replay stale après une opération destructive ;
- travail inutile sur un buffer explicitement non borné ;
- prolongation d'un stream attaché après expiration de sa rétention ;
- non-respect direct du contrat d'éviction.

Bug 2 — Le Stop source peut redevenir actif après fermeture/réouverture

Propriété

Évaluation

Gravité  **Moyenne à élevée**

Confiance **100 %**

Zone Stop source + Open + frontend

Contrat violé L'abort source est un claim one-shot et doit rester visuellement consommé

Le backend met correctement `abortRequested=true` dans le registre et `stopRequested=true` dans Activité via l'[endpoint abortSource](#).

Mais :

- [le résultat Open partagé](#) ne transporte pas l'état `abortRequested/stopRequested` ;
- le registre autorise encore un Open live pendant que la source abortée converge ;
- [le hook frontend liveSession](#) initialise systématiquement `sourceStopAvailable=true` pour un bootstrap normalPrompt ;
- si le nouveau clic reçoit `{ accepted: false }`, le hook retire seulement l'état pending et **réactive le bouton**.

Scénario défectueux

1. Stop est accepté sur le compositeur ou dans Activité.
2. Le panneau est fermé avant le terminal.
3. Activité affiche bien le Stop détaché désactivé.
4. Le panneau est rouvert pendant la convergence de l'abort.
5. Le nouveau runtime frontend oublie que le claim a déjà été consommé.
6. Le Stop normal réapparaît actif.
7. Chaque nouveau clic est refusé, puis le bouton est de nouveau activé.

Impact

- UX trompeuse ;
- répétition possible de RPC refusés ;
- perte de l'invariant visuel « Stop one-shot exact ».

Bug 3 — Une exception synchrone dans `onData` déclenche une finalisation frontend prématurée

Propriété

Évaluation

Gravité  **Moyenne à élevée**

Confiance **100 %**

Zone Stream frontend et finalisation

Propriété

Évaluation

Contrat violé `completed/transportError` et finalisation seulement après Queue A, Queue B et `commit`

Dans `le hook useAiStream`, le `catch` de `onData` :

- exécute immédiatement `setIsStreaming(false)` ;
- relance ensuite l'exception.

Mais `le bus frontend` appelle directement le listener `onData` depuis le handler de message sans convertir cette exception en callback de stream `onError`.

En parallèle, `la finalisation du contrôleur Prompt` considère tout passage à `isStreaming=false` comme une frontière permettant de publier `finalizedStreamInstanceId`.

Scénario défectueux

Une erreur synchrone peut notamment provenir :

- d'un `turnId` incohérent ;
- d'une défaillance synchrone du traitement sideband ;
- d'une exception d'entrée avant la prise en charge normale par Queue A.

Le frontend peut alors :

1. passer visuellement idle ;
2. publier `finalizedStreamInstanceId` ;
3. démasquer le compositeur ;
4. conserver encore l'identité single-flight active ;
5. ne posséder encore aucun outcome `completed` ou `transportError`.

Impact

- finalisation publiée avant les barrières prévues ;
- tentative suivante éventuellement refusée alors que l'UI paraît idle ;
- masque fallback potentiellement retiré trop tôt ;
- terminal synthétique ou métadonnées tardives susceptibles de ne plus posséder leurs cibles UI.

Bug 4 — Une soumission refusée peut retirer le masque appartenant à une autre tentative acceptée

Propriété

Évaluation

Gravité  Moyenne à élevée

Confiance $\approx 92\%$

Zone Barrière `fallback_persisted` du compositeur

Contrat violé Une tentative refusée ne doit muter aucun état appartenant à la tentative déjà acceptée

Dans `l'intégration Prompt` :

- après un `start` refusé, le callback exécute `setSubmitTxnPending(false)` ;
- dans son `catch`, il efface inconditionnellement `pendingClearAfterFallbackAckRef.current`.

Ces mutations ne vérifient pas que l'invocation courante possède la barrière.

Scénario possible

1. La tentative A est acceptée et arme sa barrière.
2. Avant le rerender protecteur, une seconde invocation atteint le même callback.
3. Le single-flight refuse B.
4. B retire le masque appartenant à A.
5. L'utilisateur peut modifier le brouillon pendant que la barrière A est toujours armée.
6. L'ack A arrive ensuite et efface le nouveau contenu.

Le déclenchement utilisateur est étroit parce que React traite normalement les événements discrets rapidement, mais le défaut de propriété transactionnelle existe bien dans le code et peut être reproduit par appel direct ou concurrence same-stack.

Conclusion de la question 3

Oui : l'implémentation possède au moins trois bugs certains et une course de robustesse supplémentaire, dont l'un viole directement l'éviction destructive et un autre peut casser la sémantique one-shot du Stop après réouverture.

4) « Quel est votre avis sur le code source et sur la spécification ? »

Avis sur la spécification

La spécification est **exceptionnellement forte sur les invariants** :

- séparation claire entre `turnId`, `sourceId`, `streamInstanceId` et tentative Open ;
- ordonnancement précis des mutations avant/après `await` ;

- distinction nette entre source, observer, lease, record, retraite, tombstone et claim ;
- prise en compte explicite des erreurs post-header, du replay, de l'ABA et des sources successives ;
- critères d'isomorphisme suffisamment détaillés pour éviter les « presque corrects » dangereux.

Son facteur de risque annoncé de **92 %** était crédible. La réalisation touche simultanément le transport, la persistance, les panneaux, React, les queues et les suppressions destructives.

Son principal défaut est sa **taille extrême** : les règles sont excellentes individuellement, mais la surface de 160 opérations planifiées — et finalement 189 fichiers touchés — rend difficile la preuve mentale qu'aucune interaction n'a été oubliée.

Les bugs trouvés sont précisément des jonctions entre deux contrats pourtant corrects séparément.

Avis sur le code source

Points très réussis

- `sourceId` est propagé proprement sur toutes les couches importantes.
- Les réservations sont réellement synchrones et placées avant les premiers `await`.
- Les guards de tentative dans Queue A et Queue B sont bien conçus.
- La séparation entre [le lifecycle source partagé](#) et [l'adaptateur source normalPrompt](#) est excellente.
- Le coordinateur de panneau est injecté et ne dépend pas directement du low-level opener.
- `preserveLocal` respecte réellement l'absence de lecture du brouillon.
- Les commentaires techniques sont nombreux, ciblés et utiles.
- La couverture de tests est largement supérieure à la moyenne.
- Les modifications du framework et du générateur Slimbk sont plus propres qu'un contournement local.
- L'utilisation d'un pending local pour le Stop détaché est meilleure que l'utilisation littérale de `disableAfterClick`, car elle permet une nouvelle tentative lorsque le RPC est réellement refusé.

Points fragiles

- [le registre liveSession](#) dépasse largement le millier de lignes et concentre beaucoup de règles de temporalité.
- [l'intégration Prompt](#) reste le point de rencontre de trop nombreuses autorités locales.
- L'autorité Stop consommée n'est pas transportée par Open, ce qui oblige le frontend à l'inventer lors d'un nouveau montage.
- L'éviction retire les index publics mais pas complètement les capacités internes détenues par les subscriptions.
- Les tests sont nombreux, mais certaines matrices testent l'état avant/après sans injecter un suffixe encore non lu, un remount après Stop ou une exception `onData`.

Conclusion de la question 4

La qualité générale est très élevée et l'architecture est sérieuse, mais la tranche n'est pas encore totalement fermée : les défauts restants sont des erreurs de propriété temporelle, pas des fonctionnalités manquantes triviales.

5) « Résumez votre rapport dans un tableau. »

Axe	Estimation	Verdict
Présence des modules et fichiers demandés	≈ 99 %	 Pratiquement complète
Contrats et typage partagés	≈ 99 %	 Très solides
Registre, réservations, époques et claims	≈ 97 %	 Éviction des leases incomplète
Source Prompt, persistance et protocole Slimbk	≈ 99 %	 Très solide
Origine exacte des webviews	100 %	 Conforme
Panneaux, Activité et cleanup	≈ 98 %	 Conforme dans les chemins normaux
Stop source et Stop détaché	≈ 90 %	 Autorité perdue lors d'un remount
Démarrage transactionnel frontend	≈ 98 %	 Correct dans les command handlers
Queue A, Queue B et finalisation	≈ 95 %	 Chemin d'exception <code>onData</code> incomplet
Barrière <code>fallback_persisted</code>	≈ 94 %	 Propriété insuffisante en soumission concurrente
Handoff direct et <code>preserveLocal</code>	100 %	 Conforme
Continuation et thinking	≈ 99 %	 Conforme
Delete et Retry destructifs	≈ 95 %	 Dépend de la correction d'éviction
Préservation EasyAgents	≈ 99 %	 Très bonne
Documentation et tests présents	≈ 97 %	 Excellente couverture, quatre trous ciblés
Préparation production globale	≈ 92–94 %	 Corrections recommandées avant clôture

6) « Concluez en une phrase. »

La tranche est très largement et intelligemment implémentée, mais elle ne doit pas être déclarée totalement conforme tant que l'éviction des leases, la persistance de l'autorité Stop après réouverture et les deux courses frontend de finalisation ne sont pas corrigées.

TL-DR

-  Aucun grand sous-système demandé n'est absent.
-  L'identité exacte, la réservation synchrone, le protocole V1, le replay, les claims, Activité, le handoff direct et `preserveLocal` sont réellement présents.
-  La couverture de tests et la documentation sont considérables.
-  Une éviction laisse une subscription déjà attachée encore autorisée à drainer son suffixe.
-  Un Stop déjà accepté peut réapparaître activé après fermeture/réouverture.
-  Une exception `onData` peut publier la finalisation frontend avant l'outcome et les queues.
-  Une soumission refusée concurrente peut retirer le masque d'une tentative qu'elle ne possède pas.
- **Verdict global : environ 96 % implémenté, 92–94 % prêt pour la production.**

7) « Si des corrections sont nécessaires, proposez un ou plusieurs ensembles de corrections. »

Ensemble A — Autorité `liveSession`, éviction et Stop exact

- Dans [le registre `liveSession`](#) :
- révoquer les leases actives avant de signaler l'éviction ;
- garantir que `isSubscriptionReplayAuthorized(...)` devient immédiatement faux ;
- couvrir l'éviction destructive et l'expiration de rétention avec un suffixe non encore consommé.
- Dans le contrat Open et [le hook frontend `liveSession`](#) :
- transporter ou dériver l'état one-shot de l'abort source ;
- empêcher le remount de réactiver un Stop déjà revendiqué ;
- ajouter le scénario Stop accepté → fermeture immédiate → réouverture avant terminal.

Ensemble B — Courses transactionnelles frontend

- Dans [le hook `stream frontend`](#) :
- remplacer le `setIsStreaming(false)` direct de `onData` par une finalisation `transportError` partagée ;
- garantir le drainage Queue A/Queue B et le commit avant publication de la fin ;
- ajouter une couverture de chunk avec mauvais `turnId` et d'erreur synchrone du sideband.
- Dans [l'intégration Prompt](#) :
- rendre la barrière de soumission explicitement propriétaire de son `streamInstanceId` ;
- interdire à une invocation refusée ou ayant échoué avant acquisition de retirer la barrière d'une autre tentative ;
- ajouter un test tentative A acceptée + tentative B refusée avant rerender.

Ensemble C — Toutes les corrections

- Appliquer les ensembles A et B ensemble.
- Ajouter les quatre tests de régression ciblés.
- Mettre à jour les documents d'architecture concernés afin que l'éviction, le Stop one-shot remount-safe et la finalisation d'erreur frontend soient explicitement décrits.
- Ne modifier que les sources génératrices lorsque les contrats Slimbk exigent une régénération.



Corriger autorité `liveSession` et Stop



Corriger les courses frontend



Corriger toutes les anomalies

</ASSISTANT>

<USER>

[Corriger toutes les anomalies]

</USER>

<ASSISTANT>

I'm thinking...

</ASSISTANT>

<ASSISTANT>



 *reading project:*  e ["Éviction et abonnements"](#),  e ["Lecture abonnement replay"](#),  e ["Autorité Stop après Open"](#),  e ["Erreur synchrone onData"](#),  e ["Barrière fallback Prompt"](#),  e ["Schéma résultat live"](#),  e ["Tests"](#)

[éviction replay](#)",  e ["Tests Stop après Open"](#),  e ["Tests erreur onData"](#),  e ["Tests barrière refus"](#),  [liveSession.programming-doc-readme.md](#),  [LiveSessionRegistry.service.ts](#) [L540-L640],  [LiveSessionRegistry.service.ts](#) [L900-L1125],  [LiveSessionRecord.type.ts](#) [L1-L260],  [LiveSessionOpenResult.type.ts](#),  [LiveSessionOpen.service.ts](#),  [usePromptLiveSession.hook.ts](#) [L1-L620],  [useAiStream.hook.ts](#) [L1-L820],  [Prompt.tsx](#) [L850-L1060],  [prompt-stream-dispatch-lock.programming-doc-readme.md](#),  [fallback-persist-and-title.programming-doc-readme.md](#),  [useAiStream.hook.ts](#) [L800-L1172],  [Prompt.tsx](#) [L560-L850],  e ["Masque après acceptation"](#),  e ["Callback Send transactionnel"](#),  e ["Callback Edit transactionnel"](#),  [LiveSessionReplay.service.ts](#) [L1-L120],  [Prompt.tsx](#) [L1140-L1295],  [LiveSessionReplay.service.test.ts](#),  [LiveSessionRegistry.service.test.ts](#) [L1-L380],
 [LiveSessionContracts.test.ts](#),
 [LiveSessionOpen.service.test.ts](#),
 [usePromptLiveSession.normalPrompt.test.tsx](#),
 [useAiStream.hook.promptSourceIdentity.test.ts](#),
 [Prompt.fallbackPersistAck.clearInput.test.tsx](#),  [PromptFallbackPersistBarrier.testHarness.tsx](#) [L1-L440],  [PromptFallbackPersistBarrier.testHarness.tsx](#) [L440-L514],  e ["Fixtures Open live frontend"](#),  e ["Fixtures Open live backend"](#),  e ["Fixtures replay replace"](#),  e ["Branches live littérales"](#),  e ["Branches live endpoint"](#),  [LiveSessionRegistry.service.ts](#) [L400-L545],  [open.endpoint.test.ts](#),  [usePromptLiveSession.hydratation.test.tsx](#) [L1-L230],  [usePromptLiveSession.handoff.test.tsx](#) [L300-L390],  [usePromptLiveSession.normalPromptIsomorphism.test.tsx](#) [L1-L140],  [LiveSessionRegistry.service.ts](#) [L620-L720]
  scanning git:  ["Toutes les réponses Open"](#),  ["Tous les parseurs Open"](#),  ["Tous les mocks Open live"](#)
  reading project:  [usePromptLiveSession.handoff.test.tsx](#) [L110-L450],  [usePromptLiveSession.normalPrompt.test.tsx](#) [L410-L525],  [LiveSessionContracts.test.ts](#) [L135-L345],  [LiveSessionOpen.service.test.ts](#) [L50-L135],  e ["Lecture abortRequested exacte"](#),  e ["Snapshot exact du run"](#),  e ["Snapshots du registre"](#),  [LiveSessionRegistry.service.ts](#) [L800-L870],  e ["Constructions du service Open"](#)
  patching file:  [LiveSessionRegistry.service.ts](#)
[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)

Copy

Review

Accept

PatchFull file

```

-      const activeLease = record.activeLeases.get(leaseId);
-
+      const activeLease = record.activeLeases.get(leaseId);
+      const leaseReplayAuthorized = activeLease !== undefined;
+
-
-      return {
-        chunks: record.bufferedChunks.slice(cursor),
-        nextCursor: record.bufferedChunks.length,
-
+      return {
+        // An evicted/released consumer must not copy or expose an unbounded stale
+        // suffix.
+        chunks: leaseReplayAuthorized
+          ? record.bufferedChunks.slice(cursor)
+          : [],
+        nextCursor: leaseReplayAuthorized
+          ? record.bufferedChunks.length
+          : cursor,
+
-
-        leaseState,
-        leaseReplayAuthorized: activeLease !== undefined,
-
+        leaseState,
+        leaseReplayAuthorized,

```

```

+
-
-    /** Await the exact source transport relinquishing persistence authority, or
resolve immediately if already gone. */
-
+
+    /** Read the current one-shot abort latch for one exact reservation or active
record. */
+    public getAbortRequestedForExactRun(
+        runRef: LiveSessionRunRef,
+    ): boolean | undefined {
+        this.sweep();
+        const runKey = buildLiveSessionRunKey(runRef);
+        const reservation = this.reservationsByRunKey.get(runKey);
+        if (reservation) {
+            return reservation.abortRequested;
+        }
+
+        const record = this.recordsByRunKey.get(runKey);
+        return record && !record.evicted ? record.abortRequested : undefined;
+    }
+
+    /** Await the exact source transport relinquishing persistence authority, or
resolve immediately if already gone. */
+
-
-    }
-    record.evicted = true;
-
+    }
+    const runKey = buildLiveSessionRunKey(record.runRef);
+
+    // Replay subscriptions retain a direct record reference after public indexes
are removed.
+    // Revoke every active lease before the wake-up so no consumer can drain a
stale suffix.
+    for (const leaseId of record.activeLeases.keys()) {
+        record.retiredLeaseIds.add(leaseId);
+    }
+    record.activeLeases.clear();
+    for (const leaseId of record.retiredLeaseIds) {
+        if (this.runKeyByKnownLeaseId.get(leaseId) === runKey) {
+            this.runKeyByKnownLeaseId.delete(leaseId);
+        }
+    }
+
+    // Complete all record and locator mutations before notifying subscription
waiters.
+    record.evicted = true;
+
-
-    record.evicted = true;
-    this.signal(record);
-
+    record.evicted = true;
+
-
-    this.clearActiveRunPointerIfExact(record.runRef);
-    const runKey = buildLiveSessionRunKey(record.runRef);
-
+    this.clearActiveRunPointerIfExact(record.runRef);
+
-
-    }

```

```

-         for (const leaseId of [
-             ...record.activeLeases.keys(),
-             ...record.retiredLeaseIds,
-         ]) {
-             if (this.runKeyByKnownLeaseId.get(leaseId) === runKey) {
-                 this.runKeyByKnownLeaseId.delete(leaseId);
-             }
-         }
+     }
+     this.signal(record);
+

```

[AI Code/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)

Copy

Review

Accept



patching file: [LiveSessionRegistry.service.test.ts](#)

[AI Code/src/extension/src/aiTools/orch/liveSession/registry/_test_/LiveSessionRegistry.service.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-
-     it('emits immutable observer-inactive wake-ups after lease removal and later source
settlement', () => {
-
+
+     it('revokes attached replay leases before destructive eviction wakes consumers', ()
=> {
+         const registry = new LiveSessionRegistryService();
+         const reservation = reserve(registry, RUN_A);
+         promote(registry, RUN_A, reservation);
+         registry.appendChunk({
+             runRef: RUN_A,
+             chunk: token(RUN_A, 'buffered-before-eviction'),
+         });
+         expect(
+             registry.prepareOpen({ runRef: RUN_A, leaseId: 'lease' }),
+         ).toBeDefined();
+         const subscription = registry.attach({
+             runRef: RUN_A,
+             leaseId: 'lease',
+         });
+
+         const claim = registry.acquireLocatorDestructionClaim(LOCATOR);
+         if (!claim) throw new Error('Expected destruction claim.');
```

registry.evictLocator(LOCATOR, claim);

```

+
+         // The subscription still owns its opaque handle, but eviction must retire its
capability
+         // before the emitted revision can wake replay and expose any remaining
buffered suffix.
+         expect(registry.readSubscription(subscription, 0)).toEqual(
+             expect.objectContaining({
+                 chunks: [],
+                 evicted: true,
+                 leaseState: 'retired',
+                 leaseReplayAuthorized: false,
+             }),
+         );
+     });
+

```

```

+         expect(
+             registry.isSubscriptionReplayAuthorized(subscription),
+         ).toBe(false);
+         expect(registry.releaseLocatorDestructionClaim(claim)).toBe(true);
+     });
+
+     it('emits immutable observer-inactive wake-ups after lease removal and later source
settlement', () => {
+

```

[AICode/src/extension/src/aiTools/orch/liveSession/registry/ test /LiveSessionRegistry.service.test.ts](#)

Copy

Review

Accept

  **patching file:**  [LiveSessionReplay.service.test.ts](#)

[AICode/src/extension/src/aiTools/orch/liveSession/replay/ test /LiveSessionReplay.service.test.ts](#)

Copy

Review

Accept

PatchFull file

```

-         await evictedConsumer.next();
-         const pendingEviction = evictedConsumer.next();
-
+         await evictedConsumer.next();
+         evicted.registry.appendChunk({
+             runRef: RUN,
+             chunk: token('must-not-drain-after-eviction'),
+         });
+
+
-         evicted.registry.evictLocator(RUN.locator, claim);
-         await expect(pendingEviction).resolves.toEqual({
-
+         evicted.registry.evictLocator(RUN.locator, claim);
+
+         // The generator is suspended immediately after yielding the first backlog
item. On resume,
+         // destructive eviction must revoke its lease before the buffered tail can be
delivered.
+         await expect(evictedConsumer.next()).resolves.toEqual({
+

```

[AICode/src/extension/src/aiTools/orch/liveSession/replay/ test /LiveSessionReplay.service.test.ts](#)

Copy

Review

Accept

  **patching file:**  [LiveSessionOpenResult.type.ts](#)

[AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionOpenResult.type.ts](#)

Copy

Review

Accept

PatchFull file

```

-         ZLiveSessionIdentity.extend({
-             kind: z.literal('live'),
-             replayMode: ZLiveSessionReplayMode,
-
+         ZLiveSessionIdentity.extend({
+             kind: z.literal('live'),
+             replayMode: ZLiveSessionReplayMode,

```

```

+         abortRequested: z.boolean(),
+
-         replayMode: ZLiveSessionReplayMode.optional(),
-
+         replayMode: ZLiveSessionReplayMode.optional(),
+         abortRequested: z.boolean().optional(),
+
-         if (value.kind === 'live') {
-             if (
-                 !hasOwn(value, 'replayMode') ||
-                 value.replayMode === undefined
-             ) {
-                 context.addIssue({
-                     code: z.ZodIssueCode.custom,
-                     path: ['replayMode'],
-                     message: 'replayMode is required for live',
-                 });
-
+             if (value.kind === 'live') {
+                 for (const key of [
+                     'replayMode',
+                     'abortRequested',
+                 ] as const) {
+                     if (!hasOwn(value, key) || value[key] === undefined) {
+                         context.addIssue({
+                             code: z.ZodIssueCode.custom,
+                             path: [key],
+                             message: `${key} is required for live`,
+                         });
+                     }
+                 }
-             }
-             } else if (hasOwn(value, 'replayMode')) {
-                 context.addIssue({
-                     code: z.ZodIssueCode.custom,
-                     path: ['replayMode'],
-                     message: 'replayMode is forbidden for terminal',
-                 });
-             }
+             } else {
+                 for (const key of [
+                     'replayMode',
+                     'abortRequested',
+                 ] as const) {
+                     if (hasOwn(value, key)) {
+                         context.addIssue({
+                             code: z.ZodIssueCode.custom,
+                             path: [key],
+                             message: `${key} is forbidden for terminal`,
+                         });
+                     }
+                 }
+             }
-         }
-         'sourceId',
-         'replayMode',
-
+         'sourceId',
+         'replayMode',
+         'abortRequested',
+

```

```
-      replayMode: ZLiveSessionReplayMode,
-    }).strict(),
-
+      replayMode: ZLiveSessionReplayMode,
+      abortRequested: z.boolean(),
+    }).strict(),
+
```

[AICode/src/sharedlib/src/types/liveSessionTypes/LiveSessionOpenResult.type.ts](#)

Copy

Review

Accept



patching file: [LiveSessionOpen.service.ts](#)

[AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts](#)

Copy

Review

Accept

PatchFull file

```
-      | 'hasExactRun'
-      | 'hasExactReservation'
-
+      | 'hasExactRun'
+      | 'hasExactReservation'
+      | 'getAbortRequestedForExactRun'
+
```

```
-      | 'hasExactReservation'
-
+      | 'hasExactReservation'
+      | 'getAbortRequestedForExactRun'
+
```

```
-      return { kind: 'not_found' };
-    }
-
+      return { kind: 'not_found' };
+    }
+    const abortRequested =
+      this.registry.getAbortRequestedForExactRun(args.runRef);
+    if (abortRequested === undefined) {
+      return { kind: 'not_found' };
+    }
+
```

```
-      replayMode: permit.replayMode,
-
+      replayMode: permit.replayMode,
+      // Read after snapshot I/O so a Stop claimed while persistence was
loading cannot be lost.
+      abortRequested,
+
```

[AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts](#)

Copy

Review

Accept



patching file: [LiveSessionOpen.service.test.ts](#)

[AICode/src/extension/src/aiTools/orch/liveSession/open/_test_/LiveSessionOpen.service.test.ts](#)

Copy

Review

Accept

PatchFull file

```
-         replayMode: 'replaceTurnAssistants',
-
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+
-
-         replayMode: 'continueAssistant',
-
+         replayMode: 'continueAssistant',
+         abortRequested: false,
+
-
-    });
-
+    });
+
+    it('returns the current abort latch when Stop is claimed during snapshot I/O',
+    async () => {
+        const registry = new LiveSessionRegistryService();
+        promote(registry, 'continueAssistant', [
+            new AssistantMessage(RUN.turnId, 'baseline'),
+        ]);
+        let resolveRead: ((value: ChatSessionFile) => void) | undefined;
+        const pendingRead = new Promise<ChatSessionFile>((resolve) => {
+            resolveRead = resolve;
+        });
+        const service = new LiveSessionOpenService({
+            registry,
+            chatReader: { loadSessionFile: async () => await pendingRead },
+        });
+
+        const openPromise = service.open({
+            runRef: RUN,
+            leaseId: 'lease-1',
+        });
+        const abortClaim = registry.claimSourceAbort(RUN);
+        if (!abortClaim) throw new Error('Expected source abort claim.');
```

[AICode/src/extension/src/aiTools/orch/liveSession/open/ test /LiveSessionOpen.service.test.ts](#)

Copy

Review

Accept



patching file: [open.endpoint.test.ts](#)

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/ test /open.endpoint.test.ts](#)

Copy

PatchFull file

Accept

(what is it?)

PatchFull file

```
-         const envelope = parseExactOpenEnvelope(response);
-
+         const envelope = parseExactOpenEnvelope(response);
+         if (
+             envelope.kind === 'live' &&
+             envelope.abortRequested
+         ) {
+             // Open is the authoritative remount boundary for the one-shot
backend abort latch.
```

```

+                                // A reopened panel must never recreate Stop after another view
already claimed it.
+                                retireSourceStopAuthority();
+                                }
+
-                                );
-                                if (result.accepted) {
-                                    // The backend claim is one-shot: once accepted, no second Stop can
target this source.
-                                    // Retire availability and finish the local handshake before enabling
Retry again.
-                                    retireSourceStopAuthority();
-                                    } else {
-                                    clearSourceStopPending();
-                                    }
-
+                                );
+                                // Acceptance consumes the one-shot claim, while a structured refusal
proves that another
+                                // owner or terminal transition already consumed it. Only transport failure
may restore Stop.
+                                retireSourceStopAuthority();
+

```

[AI Code/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)

Copy

Review

Accept

  patching file:  [usePromptLiveSession.normalPrompt.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.normalPrompt.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

-                                await firstStop;
-                                });
-                                expect(result.current.api.sourceStopPending).toBe(false);
-
+                                await firstStop;
+                                });
+                                expect(result.current.api.sourceStopPending).toBe(false);
+                                expect(result.current.api.sourceStopAvailable).toBe(false);
+
-                                resolveOpen?.({ kind: 'not_found' });
-                                });
-
+                                resolveOpen?.({ kind: 'not_found' });
+                                });
+
+                                it('keeps source Stop retired when a reopened live source already accepted abort',
async () => {
+                                    const startLiveObserver = jest.fn<() => AiStreamStartResult>(() => ({
+                                        accepted: true,
+                                        runRef: bootstrap.runRef,
+                                        streamInstanceId: 'stream-reopened',
+                                    }));
+                                    callLocalApiMock.mockResolvedValueOnce({
+                                        kind: 'live',
+                                        session: {
+                                            meta: {

```

```

+         formatVersionNumber: 1,
+         sessionId: locator.sessionId,
+         title: 'Chat',
+         createdAt: 1,
+         updatedAt: 2,
+         firstSendState: 'postSend',
+       },
+       messages: [],
+     },
+     threadId: locator.sessionId,
+     turnId: bootstrap.runRef.turnId,
+     sourceId: bootstrap.runRef.sourceId,
+     replayMode: 'continueAssistant',
+     abortRequested: true,
+   });
+
+   const { result } = renderHook(() =>
+     useNormalPromptLiveSessionHarness(startLiveObserver),
+   );
+
+   await act(async () => {
+     await Promise.resolve();
+     await Promise.resolve();
+   });
+   expect(result.current.mode).toBe('liveSessionStreaming');
+   expect(result.current.api.sourceStopAvailable).toBe(false);
+   await expect(result.current.api.stopSource()).resolves.toBe(false);
+   expect(callLocalApiMock).toHaveBeenCalledTimes(1);
+ });
+

```

```

-         replayMode: 'continueAssistant',
-       });
-

```

```

+         replayMode: 'continueAssistant',
+         abortRequested: false,
+       });
+

```

```

-         replayMode: 'continueAssistant',
-       })
-

```

```

+         replayMode: 'continueAssistant',
+         abortRequested: false,
+       })
+

```

[AICode/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.normalPrompt.test.tsx](#)

Copy

Review

Accept

  patching file: [usePromptLiveSession.hydraton.test.tsx](#)

[AICode/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.hydraton.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

-         sourceId: 'source',
-         replayMode: 'replaceTurnAssistants',
-
+         sourceId: 'source',
+         replayMode: 'replaceTurnAssistants',
+

```

```
+      abortRequested: false,  
+
```

```
-      sourceId: 'different-source',  
-      replayMode: 'replaceTurnAssistants',  
-
```

```
+      sourceId: 'different-source',  
+      replayMode: 'replaceTurnAssistants',  
+      abortRequested: false,  
+
```

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.hydration.test.tsx](#)

Copy

Review

Accept

  *patching file:*  [usePromptLiveSession.handoff.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.handoff.test.tsx](#)

Error

([what is it?](#))

  *patching file:*  [usePromptLiveSession.normalPromptIsomorphism.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.normalPromptIsomorphism.test.tsx](#)

Copy

Review

Accept

PatchFull file

```
-      replayMode,  
-  
+      replayMode,  
+      abortRequested: false,  
+
```

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.normalPromptIsomorphism.test.tsx](#)

Copy

Review

Accept

  *patching file:*  [useAiStream.hook.ts](#)

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy

Review

Accept

PatchFull file

```
-  
- /** Compare frontend-only attempt identity without trusting a logical turnId reused by  
retries. */  
-  
+  
+ /** One exact terminal request shared by completion, transport errors, and synchronous  
listener failures. */  
+ type ActiveAiStreamFinalizationRequest = {  
+   readonly identity: ActiveAiStreamIdentity;  
+   readonly transportFailureDetails: string[];  
+ };  
+  
+ /** Compare frontend-only attempt identity without trusting a logical turnId reused by  
retries. */  
+
```

```

-         undefined,
-     );
-
+         undefined,
+     );
+     const finalizationRequestRef = useRef<
+         ActiveAiStreamFinalizationRequest | undefined
+     >(undefined);
+

```

```

- /**
-  * Reset every visible/live-only buffer of the hook without starting a new stream.
-
+ /**
+  * Request the unique terminal pipeline for one accepted attempt.
+  *
+  * The request is latched synchronously so late data cannot enter Queue A after
+  * completion or a
+  * fatal listener failure. A later transport error may still upgrade an in-flight
+  * completion to
+  * `transportError`, but every visible state mutation remains behind Queue A, Queue
+  * B, and commit.
+  */
+     const requestAttemptFinalization = useCallback(
+         (
+             identity: ActiveAiStreamIdentity,
+             transportFailureDetails?: string,
+         ): void => {
+             if (
+                 !isSameStreamAttempt(
+                     activeIdentityRef.current,
+                     identity.attemptIdentity,
+                 )
+             )
+                 return;
+
+             const existingRequest = finalizationRequestRef.current;
+             if (existingRequest) {
+                 if (
+                     transportFailureDetails &&
+                     isSameStreamAttempt(
+                         existingRequest.identity,
+                         identity.attemptIdentity,
+                     )
+                 ) {
+                     // Preserve every transport signal that arrived while the exact
+                     // queue drain was in flight.
+                     existingRequest.transportFailureDetails.push(
+                         transportFailureDetails,
+                     );
+                 }
+                 return;
+             }
+
+             const request: ActiveAiStreamFinalizationRequest = {
+                 identity,
+                 transportFailureDetails: transportFailureDetails
+                     ? [transportFailureDetails]
+                     : [],
+             };
+             finalizationRequestRef.current = request;
+
+             void (async () => {
+                 const queueFailures = await drainAttemptQueues(

```

```

+         identity.attemptIdentity,
+     );
+     let syntheticTransportErrorQueued = false;
+     try {
+         if (
+             !isSameStreamAttempt(
+                 activeIdentityRef.current,
+                 identity.attemptIdentity,
+             )
+         )
+             return;
+
+         if (
+             !terminalDataSeenRef.current &&
+             (request.transportFailureDetails.length > 0 ||
+              queueFailures.length > 0)
+         ) {
+             // The failed item was removed by its queue. Consume retained
+             // routing one canonical terminal through the recovered exact-
+             // failures before
+             // attempt pipeline.
+
+             forgetAttemptQueueFailures(identity.attemptIdentity);
+             enqueueSyntheticTransportError({
+                 identity,
+                 details: [
+                     ...request.transportFailureDetails,
+                     ...(queueFailures.length > 0
+                      ? [formatStreamFailures(queueFailures)]
+                      : []),
+                 ].join('\n'),
+             });
+             syntheticTransportErrorQueued = true;
+             queueFailures.push(
+                 ...(await drainAttemptQueues(
+                     identity.attemptIdentity,
+                 )),
+             );
+         }
+
+         await streamQueue.waitForCommit();
+
+         if (
+             !syntheticTransportErrorQueued &&
+             !terminalDataSeenRef.current &&
+             request.transportFailureDetails.length > 0
+         ) {
+             // Completion may begin first and receive a transport error
+             // barrier is pending. Re-check after that await so the
+             // receives one canonical terminal before the final idle
+             // publication.
+
+             forgetAttemptQueueFailures(identity.attemptIdentity);
+             enqueueSyntheticTransportError({
+                 identity,
+                 details: request.transportFailureDetails.join(
+                     '\n',
+                 ),
+             });
+             queueFailures.push(
+                 ...(await drainAttemptQueues(
+                     identity.attemptIdentity,
+                 )),
+             );
+         }
+     }
+ }

```



```

+         finalizationRequestRef.current = undefined;
+
-         if (activeIdentityRef.current) return { accepted: false };
-
+         if (activeIdentityRef.current) return { accepted: false };
+         finalizationRequestRef.current = undefined;
+
-         return;
-         try {
-
+         return;
+         const finalizationRequest = finalizationRequestRef.current;
+         if (
+             finalizationRequest &&
+             isSameStreamAttempt(
+                 finalizationRequest.identity,
+                 identityAtEntry.attemptIdentity,
+             )
+         )
+             return;
+         try {
-         } catch (err) {
-             // Surface the failure explicitly and guarantee UI unblocks from
"streaming" state.
-             // This avoids leaving isStreaming stuck to true if the handler
fails during abort window.
-             console.error('[useAiStream.onData] handler failed:', err);
-             if (
-                 isSameStreamAttempt(
-                     activeIdentityRef.current,
-                     identityAtEntry.attemptIdentity,
-                 )
-             ) {
-                 setIsStreaming(false);
-             }
-             throw err instanceof Error ? err : new Error(String(err));
+         } catch (err) {
+             // A synchronous listener failure is a transport failure, not an
early UI-idle boundary.
+             // Route it through the same exact queue/commit finalizer instead
of throwing through the bus.
+             const listenerFailure = normalizeStreamFailure(err);
+             const listenerFailureDetails = `[useAiStream.onData] handler
failed: ${
+                 listenerFailure.stack ?? listenerFailure.message
+             }`;
+             console.error(
+                 '[useAiStream.onData] handler failed:',
+                 listenerFailure,
+             );
+             requestAttemptFinalization(
+                 identityAtEntry,
+                 listenerFailureDetails,
+             );
-         return;
-         // Finalize only after the queue drained and one commit turn has
passed.
-         void (async () => {

```

```

const queueFailures = await drainAttemptQueues(
  identityAtEntry.attemptIdentity,
);
if (
  queueFailures.length > 0 &&
  !terminalDataSeenRef.current &&
  isSameStreamAttempt(
    activeIdentityRef.current,
    identityAtEntry.attemptIdentity,
  )
) {
  // A failed queue item is already removed. Consume its retained
  failure, then route one
  // canonical terminal through the same queues without
  duplicating prior terminal data.
  forgetAttemptQueueFailures(
    identityAtEntry.attemptIdentity,
  );
  enqueueSyntheticTransportError({
    identity: identityAtEntry,
    details: formatStreamFailures(queueFailures),
  });
  queueFailures.push(
    ...(await drainAttemptQueues(
      identityAtEntry.attemptIdentity,
    )),
  );
}
try {
  await streamQueue.waitForCommit();
} catch (error) {
  queueFailures.push(normalizeStreamFailure(error));
} finally {
  if (
    isSameStreamAttempt(
      activeIdentityRef.current,
      identityAtEntry.attemptIdentity,
    )
  ) {
    // Only the still-owning attempt may reset global lock/gate
    state and publish completion.
    forgetAttemptQueueFailures(
      identityAtEntry.attemptIdentity,
    );
    PromptStreamLockStore.reset();
    dispatchGateQueue.resetInternalState();
    if (queueFailures.length > 0) {
      console.error(
        '[useAiStream.onComplete] queue finalization
failed:',
        new AggregateError(
          queueFailures,
          'Stream completion queues failed for the exact
accepted attempt.',
        ),
      );
    }
    setOutcome(
      queueFailures.length > 0
        ? 'transportError'
        : 'completed',
    );
    updateTransportStopAvailable(false);
  }
}

```

```

-                 setIsStreaming(false);
-                 activeIdentityRef.current = undefined;
-                 activeStreamRef.current = undefined;
-             }
-         }
-     })();
+
+     return;
+     requestAttemptFinalization(identityAtEntry);
+
-     console.error(errMsg);
-
-     // Wait until all items are applied and one commit turn passed, then
-     flip streaming off.
-     void (async () => {
-         // Drain terminal data already accepted by Queue A/B before
-         deciding whether a synthetic error is needed.
-         const queueFailures = await drainAttemptQueues(
-             identityAtEntry.attemptIdentity,
-         );
-         try {
-             if (
-                 !isSameStreamAttempt(
-                     activeIdentityRef.current,
-                     identityAtEntry.attemptIdentity,
-                 )
-             )
-                 return;
-             if (!terminalDataSeenRef.current) {
-                 // Queue failures are consumed before retrying the
-                 canonical transport terminal.
-                 forgetAttemptQueueFailures(
-                     identityAtEntry.attemptIdentity,
-                 );
-                 enqueueSyntheticTransportError({
-                     identity: identityAtEntry,
-                     details: errMsg,
-                 });
-                 queueFailures.push(
-                     ...(await drainAttemptQueues(
-                         identityAtEntry.attemptIdentity,
-                     )),
-                 );
-             }
-             await streamQueue.waitForCommit();
-         } catch (error) {
-             queueFailures.push(normalizeStreamFailure(error));
-         } finally {
-             if (
-                 isSameStreamAttempt(
-                     activeIdentityRef.current,
-                     identityAtEntry.attemptIdentity,
-                 )
-             ) {
-                 // Stale errors cannot reset successor queues or append a
-                 second terminal.
-                 forgetAttemptQueueFailures(
-                     identityAtEntry.attemptIdentity,
-                 );
-                 PromptStreamLockStore.reset();
-                 dispatchGateQueue.resetInternalState();
-                 if (queueFailures.length > 0) {
-                     console.error(

```

```

-                                     '[useAiStream.onError] queue finalization failed:',
-                                     new AggregateError(
-                                         queueFailures,
-                                         'Transport-error queues failed for the exact
accepted attempt.',
-                                     ),
-                                     );
-                                     }
-                                     setOutcome('transportError');
-                                     updateTransportStopAvailable(false);
-                                     setIsStreaming(false);
-                                     activeIdentityRef.current = undefined;
-                                     activeStreamRef.current = undefined;
-                                     }
-                                     }
-                                     }())();
+                                     console.error(errMsg);
+                                     requestAttemptFinalization(
+                                         identityAtEntry,
+                                         errMsg,
+                                     );
+

```

```

-                                     dispatchGateQueue,
-                                     drainAttemptQueues,
-                                     enqueueSyntheticTransportError,
-                                     forgetAttemptQueueFailures,
-                                     listenerAttemptIdentity,

```

```

+                                     listenerAttemptIdentity,
+

```

```

-                                     listenerAttemptIdentity,
-                                     onRawLiveReceived,
-
+                                     listenerAttemptIdentity,
+                                     onRawLiveReceived,
+                                     requestAttemptFinalization,
+

```

```

-                                     requestAttemptFinalization,
-                                     streamQueue,
-                                     updateTransportStopAvailable,

```

```

+                                     requestAttemptFinalization,
+                                     streamQueue,
+

```

[AI Code/src/frontend/src/messageBus/useAiStream.hook.ts](#)

Copy

Review

Accept



patching file: [useAiStream.hook.promptSourceIdentity.test.ts](#)

[AI Code/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy

Review

Accept

PatchFull file

```

- import type { AiStreamedChunk } from
- '../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
-

```

```

+ import type { AiStreamedChunk } from
+ '../sharedlib/types/aiTypes/AiStreamedChunk.type.js';
+ import { AiStopErrorReason } from
+ '../sharedlib/types/aiTypes/AiStopErrorReason.type.js';
+
- let queueRunMerged: QueueRunMerged | undefined;
-
+ let queueRunMerged: QueueRunMerged | undefined;
+ const streamQueueWhenIdleMock = jest.fn<
+   (attemptIdentity: AiStreamAttemptIdentity) => Promise<void>
+ >();
+
-   },
-   whenIdle: async () => {},
-
+   },
+   whenIdle: streamQueueWhenIdleMock,
+
-   callLocalApiMock.mockResolvedValue({ accepted: true });
-
+   callLocalApiMock.mockResolvedValue({ accepted: true });
+   streamQueueWhenIdleMock.mockReset();
+   streamQueueWhenIdleMock.mockResolvedValue(undefined);
+
-
-   it('restores the exact Stop capability after refusal or RPC failure while the
attempt remains active', async () => {
-
+
+   it('finalizes a synchronous onData failure only after the exact queue barrier
drains', async () => {
+     await act(async () => {
+       root.render(React.createElement(Harness));
+     });
+     const api = harnessApiRef.current;
+     if (!api) throw new Error('HarnessApi not initialized');
+
+     let startResult: AiStreamStartResult | undefined;
+     act(() => {
+       startResult = api.start({
+         kind: 'promptQuery',
+         turnId: 'turn-current',
+         storageScopeRelPath: '',
+         aiQuery: { type: 'prompt', promptValue: 'Validate frames' },
+         aiQueryOptions: {},
+       });
+     });
+     if (!startResult?.accepted) {
+       throw new Error('Expected one accepted prompt source.');
```

```

+         const handlers = handlersByChannel.get('proxylocalapi:aiStream');
+         if (!handlers) throw new Error('Prompt stream handlers missing.');
```

idle before queue barriers.

```

+         act(() => {
+             // A mismatched turn is a fatal listener input, but it must not publish
+             handlers.onData({
+                 turnId: 'turn-stale',
+                 type: 'token',
+                 value: 'wrong turn',
+             });
+         });
+         expect(harnessApiRef.current?.isStreaming).toBe(true);
+         expect(harnessApiRef.current?.outcome).toBeUndefined();
+
+         await act(async () => {
+             resolveQueueIdle?();
+             // One macrotask boundary drains the complete async finalization chain and
+             its React commit.
+             await new Promise<void>((resolve) => window.setTimeout(resolve, 0));
+         });
+
+         expect(harnessApiRef.current?.isStreaming).toBe(false);
+         expect(harnessApiRef.current?.outcome).toBe('transportError');
+         expect(queuedChunks).toContainEqual({
+             chunk: {
+                 turnId: 'turn-current',
+                 type: 'meta',
+                 value: {
+                     metaType: 'error',
+                     value: expect.objectContaining({
+                         reason: AiStopErrorReason.SYSTEM_ERROR,
+                     }),
+                 },
+             },
+             attemptIdentity: {
+                 sourceId: startResult.runRef.sourceId,
+                 streamInstanceId: startResult.streamInstanceId,
+             },
+         });
+     });
+
+     it('restores the exact Stop capability after refusal or RPC failure while the
+     attempt remains active', async () => {
```

[AI Code/src/frontend/src/messageBus/_test_/useAiStream.hook.promptSourceIdentity.test.ts](#)

Copy

Review

Accept

  **patching file:** [Prompt.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/Prompt.tsx](#)

Copy

Review

Accept

PatchFull file

```

-         const attachmentsSnapshot = attachments;
-
+         const attachmentsSnapshot = attachments;
+         let ownedBarrier:
+         | {
+             turnId: string;
```

```

+             streamInstanceId: string;
+         }
+         | undefined;
+
-         if (!result.accepted) {
-             setSubmitTxnPending(false);
-
+         if (!result.accepted) {
+             // A refused invocation owns no barrier and must not
unmask another accepted attempt.
+
-         }
-         pendingClearAfterFallbackAckRef.current = {
-
+         }
+         ownedBarrier = {
+
-         };
-
+         };
+         pendingClearAfterFallbackAckRef.current =
+             ownedBarrier;
+
-     } catch (e) {
-         // If the send failed before the backend could ack fallback
persistence,
-         // keep the prompt in the UI and disarm the clear barrier.
-         pendingClearAfterFallbackAckRef.current = undefined;
-         // UI-only: revert masking/lock on failure.
-         setSubmitTxnPending(false);
-
+     } catch (e) {
+         const pendingBarrier =
+             pendingClearAfterFallbackAckRef.current;
+         if (
+             ownedBarrier &&
+             pendingBarrier?.turnId ===
+                 ownedBarrier.turnId &&
+             pendingBarrier.streamInstanceId ===
+                 ownedBarrier.streamInstanceId
+         ) {
+             // Only the invocation that installed this exact
barrier may remove its mask.
+             pendingClearAfterFallbackAckRef.current =
+                 undefined;
+             setSubmitTxnPending(false);
+         }
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/Prompt.tsx](#)

Copy

Review

Accept



patching file: [Prompt.fallbackPersistAck.clearInput.test.tsx](#)

[AI Code/src/frontend/src/app/prompt/Prompt/ test /Prompt.fallbackPersistAck.clearInput.test.tsx](#)

Copy

Review

Accept

PatchFull file

```

-     focusAtStartMock,
-
+     focusAtStartMock,
+     handleSendMock,
+
-   });
- });
-
+   });
+
+   it('keeps the accepted barrier masked when a same-stack follow-up start is
refused', async () => {
+     resetPromptFallbackPersistBarrierHarness({
+       promptDraft: 'keep the accepted prompt',
+       attachments: [PROBLEMS_ATTACHMENT],
+       nextTurnId: 'next-turn',
+       messages: [],
+     });
+     const rendered = renderPromptFallbackPersistBarrierHarness();
+     await waitFor(() => {
+       expect(screen.getByTestId('promptInput')).toHaveValue(
+         'keep the accepted prompt',
+       );
+     });
+
+     const sendButton = screen.getByTestId('sendPrompt');
+     act(() => {
+       // The first click owns the accepted barrier. The second click is
deliberately batched
+       // before React can protect the surface with its rendered disabled state.
+       sendButton.click();
+       handleSendMock.mockReturnValueOnce({ accepted: false });
+       sendButton.click();
+     });
+
+     expect(handleSendMock).toHaveBeenCalledTimes(2);
+     expect(screen.getByTestId('promptInput')).toBeDisabled();
+
+     await act(async () => {
+       setPromptFinalizedStreamInstanceId(PROMPT_BARRIER_SEND_STREAM_ID);
+       rerenderPromptFallbackPersistBarrierHarness(rendered);
+     });
+     await waitFor(() => {
+       expect(screen.getByTestId('promptInput')).not.toBeDisabled();
+     });
+     expect(screen.getByTestId('promptInput')).toHaveValue(
+       'keep the accepted prompt',
+     );
+     expect(screen.getByTestId('attachmentCount')).toHaveTextContent('1');
+   });
+ });
+

```

[AI Code/src/frontend/src/app/prompt/Prompt/ test /Prompt.fallbackPersistAck.clearInput.test.tsx](#)

Copy

Review

Accept



 reading project:
 
[LiveSessionContracts.test.ts](#) [L145-L340],
 
[usePromptLiveSession.handoff.test.tsx](#) [L125-L435],
 
[usePromptLiveSession.normalPrompt.test.tsx](#) [L150-L250],
 
[usePromptLiveSession.normalPrompt.test.tsx](#) [L420-L515]



 patching file:
 
[LiveSessionContracts.test.ts](#)

Error (what is it?)

Error (what is it?)

Error ([what is it?](#))

Error (what is it?)

Copy
Review
Accept
PatchFull file

```
-      sourceId: 'source-1',
```

```

-         replayMode: 'replaceTurnAssistants',
-
+         sourceId: 'source-1',
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+
-     ).toBe(false);
-     expect(
-         ZLiveSessionOpenResultTransport.safeParse({
-             kind: 'terminal',
-
+     ).toBe(false);
+     expect(
+         ZLiveSessionOpenResultTransport.safeParse({
+             kind: 'live',
+             session,
+             threadId: SESSION_ID,
+             turnId: 'turn-1',
+             sourceId: 'source-1',
+             replayMode: 'replaceTurnAssistants',
+         }).success,
+     ).toBe(false);
+     expect(
+         ZLiveSessionOpenResultTransport.safeParse({
+             kind: 'live',
+             session,
+             threadId: SESSION_ID,
+             turnId: 'turn-1',
+             sourceId: 'source-1',
+             replayMode: 'replaceTurnAssistants',
+             abortRequested: undefined,
+         }).success,
+     ).toBe(false);
+     expect(
+         ZLiveSessionOpenResultTransport.safeParse({
+             kind: 'terminal',
+
-         replayMode: undefined,
-     }).success,
-     ).toBe(false);
-
+         replayMode: undefined,
+     }).success,
+     ).toBe(false);
+     expect(
+         ZLiveSessionOpenResultTransport.safeParse({
+             kind: 'terminal',
+             session,
+             threadId: SESSION_ID,
+             turnId: 'turn-1',
+             sourceId: 'source-1',
+             abortRequested: undefined,
+         }).success,
+     ).toBe(false);
+
-         replayMode: 'continueAssistant',
-     }).success,
-
+         replayMode: 'continueAssistant',
+         abortRequested: true,
+     }).success,
+

```

[AICode/src/sharedlib/src/types/liveSessionTypes/ test /LiveSessionContracts.test.ts](#)

Copy

Review

Accept

  patching file:  [usePromptLiveSession.handoff.test.tsx](#)

[AICode/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.handoff.test.tsx](#)

Copy

Review

Accept

PatchFull file

```
-         sourceId: 'source',
-         replayMode: 'replaceTurnAssistants',
-
+         sourceId: 'source',
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+
-
-         replayMode: 'replaceTurnAssistants',
-     });
-     const { result, rerender } = renderHook(
-       (signals: HandoffSignals) =>
-         useHandoffHarness(signals, easyAgentsBootstrap),
-       { initialProps: buildInitialHandoffSignals() },
-     );
-     await waitFor(() =>
-       expect(result.current.mode).toBe('liveSessionStreaming'),
-     );
-
-     await act(async () => {
-       rerender({
-         terminalDataSeen: false,
-
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+       });
+     const { result, rerender } = renderHook(
+       (signals: HandoffSignals) =>
+         useHandoffHarness(signals, easyAgentsBootstrap),
+       { initialProps: buildInitialHandoffSignals() },
+     );
+     await waitFor(() =>
+       expect(result.current.mode).toBe('liveSessionStreaming'),
+     );
+
+     await act(async () => {
+       rerender({
+         terminalDataSeen: false,
+
-         sourceId: easyAgentsBootstrap.runRef.sourceId,
-         replayMode: 'replaceTurnAssistants',
-
+         sourceId: easyAgentsBootstrap.runRef.sourceId,
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+
-
-         replayMode: 'replaceTurnAssistants',
-       });
-     }
```

```
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+     });
+
```

```
-         sourceId: 'source',
-         replayMode: 'replaceTurnAssistants',
-
+         sourceId: 'source',
+         replayMode: 'replaceTurnAssistants',
+         abortRequested: false,
+
```

[AI Code/src/frontend/src/app/prompt/liveSession/core/ test /usePromptLiveSession.handoff.test.tsx](#)

Copy

Review

Accept

  patching file:  [liveSession.programming-doc-readme.md](#)

[AI Code/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

- Handoff retention is fixed once for 30 seconds and is never extended.
- Handoff retention is fixed once for 30 seconds and is never extended.
- Destructive or retention eviction retires every active lease before waking replay waiters. Opaque subscriptions may retain their record reference, but authorization is already false and no buffered suffix may drain after the eviction boundary.

Live Open acquires the exact opening lease before reading persistence, removes only assistant messages of the requested active turn, and never resets Orch memory. Missing or failed snapshots do not implicitly release the opening lease.

Live Open acquires the exact opening lease before reading persistence, removes only assistant messages of the requested active turn, and never resets Orch memory. Missing or failed snapshots do not implicitly release the opening lease.

The live Open branch also carries the current exact `abortRequested` latch. It is read after persistence

I/O and permit revalidation, so a Stop claimed while the snapshot was loading is visible to the reopened

panel. Normal Prompt uses that latch to keep its unique composer Stop permanently retired across remounts.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. A refused or failed frontend abort restores the exact still-active Stop capability and releases the local abort handshake so the operator may retry. The detached Activity Stop follows the same rule with local pending state instead of an irreversible click disable. When rendering already became idle but the transport capability remains stoppable, the handshake stays armed during the RPC and closes on its accepted response because no later `isStreaming` transition exists. A normal-Prompt source Stop latch is also released when replay converges to a stable failure, because that surface must keep Retry usable after a pre-baseline source retirement. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive

wake-up again so deferred panel cleanup is retried after the final authority transition. Exact sound state is retired for every source-handler finalization, including `stale`, because the full runRef prevents cleanup from touching a successor source.

Abort is claimed atomically for either an active observer lease or an exact source-owned reservation/record. Claiming synchronously latches `abortRequested`; the returned callback must be invoked without an intervening `await`. The normal-Prompt direct source route permanently retires Stop after either an accepted claim or a structured backend refusal; only a transport exception restores it for retry. Generic active-transport routes retain their existing same-attempt retry policy. The detached Activity Stop keeps local pending state and restores it only when its exact RPC is refused or fails; an accepted claim updates the shared Activity latch before abort. When rendering already became idle but the transport capability remains stoppable, the handshake stays armed during the RPC and closes on its accepted response because no later `isStreaming` transition exists. A normal-Prompt source Stop latch is also released when replay converges to a stable failure, because that surface must keep Retry usable after a pre-baseline source retirement. Reservation snapshots expose their real source/observer ownership so panel coordination and Activity rehydration never reinterpret an EasyAgents reservation as a detached normal Prompt source. The registry emits deeply immutable advisory wake-ups only after complete observer-inactive and destruction-claim-release mutations, and isolates listener failures. When the last observer lease disappeared before EasyAgents completed its business handoff, the handoff emits the observer-inactive wake-up again so deferred panel cleanup is retried after the final authority transition. Exact sound state is retired for every source-handler finalization, including `stale`, because the full runRef prevents cleanup from touching a successor source.

[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)

Copy

Review

Accept

  patching file:  [prompt-stream-dispatch-lock.programming-doc-readme.md](#)

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Error

([what is it?](#))

  patching file:  [fallback-persist-and-title.programming-doc-readme.md](#)

[AICode/src/extension/src/aiTools/orch/ask/stream/core/fallback-persist-and-title.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

The frontend arms the clear barrier only after synchronous stream acceptance and correlates it with `{ turnId, streamInstanceId }`. If the accepted attempt finalizes without receiving its matching `fallback\_persisted` ack, Prompt removes the transactional mask and restores focus but preserves prompt text, attachments, `nextTurnId`, edit state, and maximized state. Finalization is published for every accepted stream instance, including a transport that completed without visible token, thinking, or meta data, so an empty failure path cannot leave the composer masked indefinitely. A refused start creates no scaffold and arms no barrier.

The frontend arms the clear barrier only after synchronous stream acceptance and correlates it with `{ turnId, streamInstanceId }`. If the accepted attempt finalizes without receiving its matching `fallback\_persisted` ack, Prompt removes the transactional mask and restores focus but preserves prompt text, attachments, `nextTurnId`, edit state, and maximized state. Finalization is published for every accepted stream instance, including a transport that completed without visible token, thinking, or meta data, so an empty failure path cannot leave the composer masked indefinitely. A refused start creates no scaffold and arms no barrier.

Barrier removal is ownership-checked with the same `{ turnId, streamInstanceId }`. A second same-stack

submission that is refused owns no barrier and cannot unmask, clear, or replace the transaction installed by the already-accepted attempt.

[AICode/src/extension/src/aiTools/orch/ask/stream/core/fallback-persist-and-title.programming-doc-readme.md](#)

Copy

Review

Accept



reading project: [prompt-stream-dispatch-lock.programming-doc-readme.md](#) [L95-L145]



patching file: [prompt-stream-dispatch-lock.programming-doc-readme.md](#)

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept

PatchFull file

The reset/finalization boundary is guarded by ``sourceId + streamInstanceId``. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation after ``await``, or Queue A ``runMerged`` item from an older accepted stream cannot enter gating, acquire a lock, reset Queue B, or mutate the successor stream. Queue A and Queue B retain their first downstream failure per exact attempt while recovering their internal scheduling tails; ``whenIdle(attemptIdentity)`` rejects with that retained failure instead of silently reporting success. The stream owner consumes and forgets the failure only after publishing the attempt outcome. ``completed`` and ``transportError`` are published only after Queue A idle, Queue B idle, and the final React commit barrier. Any queue or commit failure forces ``transportError``; if no terminal data was consumed, one canonical error terminal is retried through the recovered queues. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

The reset/finalization boundary is guarded by ``sourceId + streamInstanceId``. Each mounted transport listener freezes that identity in its callback object, so an old bus callback cannot relabel its frames with the current stream ref. A stale callback, continuation after ``await``, or Queue A ``runMerged`` item from an older accepted stream cannot enter gating, acquire a lock, reset Queue B, or mutate the successor stream. Queue A and Queue B retain their first downstream failure per exact attempt while recovering their internal scheduling tails; ``whenIdle(attemptIdentity)`` rejects with that retained failure instead of silently reporting success. The stream owner consumes and forgets the failure only after publishing the attempt outcome. ``completed`` and ``transportError`` are published only after Queue A idle, Queue B idle, and the final React commit barrier. Any queue or commit failure forces ``transportError``; if no terminal data was consumed, one canonical error terminal is retried through the recovered queues. If terminal data was already consumed before transport failure, no second error meta or suffix is injected.

A synchronous listener failure, including an exact-turn validation failure before Queue A enqueue, enters the same attempt finalization request. It latches the attempt before accepting any later data, never flips

``isStreaming`` directly, and publishes ``transportError`` only after Queue A, Queue B, and the final commit

barrier. Completion and error callbacks share this one request, so a transport error arriving during an

in-flight completion upgrades the outcome without creating a second terminal pipeline.

[AICode/src/frontend/src/messageBus/prompt-stream-dispatch-lock.programming-doc-readme.md](#)

Copy

Review

Accept



scanning git: ["Réponses Open sans latch"](#), ["Toutes les latches Open"](#), ["Modifications actuelles"](#), ["Anomalies du patch"](#)

  reading project:  [useAiStream.hook.ts](#) [L70-L150],  [useAiStream.hook.ts](#) [L610-L870],  [useAiStream.hook.ts](#) [L900-L1100],  [LiveSessionOpenResult.type.ts](#) [L1-L180],  [LiveSessionRegistry.service.ts](#) [L800-L850],  [LiveSessionRegistry.service.ts](#) [L1060-L1115],  [LiveSessionOpen.service.ts](#) [L1-L155],  [usePromptLiveSession.hook.ts](#) [L225-L310],  [usePromptLiveSession.hook.ts](#) [L500-L555],  [Prompt.tsx](#) [L1170-L1245],  [useAiStream.hook.ts](#) [L1030-L1145],  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L1-L360],  [useAiStream.hook.promptSourceIdentity.test.ts](#) [L360-L438]

  patching file:  [usePromptLiveSession.hook.ts](#)
[AICode/src/frontend/src/app/prompt/liveSession/core/usePromptLiveSession.hook.ts](#)
Grouped
([what is it?](#))

  reading project:  [LiveSessionContracts.test.ts](#) [L145-L340],  [usePromptLiveSession.handoff.test.tsx](#) [L125-L435],  [Prompt.fallbackPersistAck.clearInput.test.tsx](#) [L1-L220],  [LiveSessionOpen.service.test.ts](#) [L50-L175],  [LiveSessionReplay.service.test.ts](#) [L70-L125],  [LiveSessionContracts.test.ts](#) [L330-L375],  [usePromptLiveSession.handoff.test.tsx](#) [L405-L435],  [usePromptLiveSession.handoff.test.tsx](#) [L435-L455]

  patching file:  [LiveSessionRegistry.service.ts](#)
[AICode/src/extension/src/aiTools/orch/liveSession/registry/LiveSessionRegistry.service.ts](#)
Grouped
([what is it?](#))

  patching file:  [LiveSessionRegistry.service.test.ts](#)
[AICode/src/extension/src/aiTools/orch/liveSession/registry/ test /LiveSessionRegistry.service.test.ts](#)
Grouped
([what is it?](#))

  patching file:  [useAiStream.hook.ts](#)
[AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)
Grouped
([what is it?](#))

  reading project:  [usePromptLiveSession.hydration.test.tsx](#) [L115-L230]
  patching file:  [useAiStream.hook.ts](#)
[AICode/src/frontend/src/messageBus/useAiStream.hook.ts](#)
Grouped
([what is it?](#))

  reading project:  [liveSession.programming-doc-readme.md](#) [L70-L105]
  patching file:  [liveSession.programming-doc-readme.md](#)
[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)
Grouped
([what is it?](#))

  reading project:  [SidebarActivityQuery.tsx](#) [L1-L150]
  patching file:  [liveSession.programming-doc-readme.md](#)
[AICode/src/extension/src/aiTools/orch/liveSession/liveSession.programming-doc-readme.md](#)
Grouped
([what is it?](#))

  patching file:  [open.endpoint.test.ts](#)
[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/ test /open.endpoint.test.ts](#)
Grouped
([what is it?](#))

  patching file:  [useAiStream.hook.promptSourceIdentity.test.ts](#)
[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)
Grouped
([what is it?](#))

  reading project:  [open.endpoint.test.ts](#) [L30-L100],  [open.endpoint.test.ts](#) [L100-L125]
  scanning recent files
  reading project:  [useAiStream.hook.ts](#) [L645-L805],  [useAiStream.hook.ts](#) [L1060-L1165],  [LiveSessionRegistry.service.test.ts](#) [L280-L355],  [usePromptLiveSession.normalPromptIsomorphism.test.tsx](#) [L90-L120],  [fallback-persist-and-title.programming-doc-readme.md](#) [L45-L75],  [prompt-stream-dispatch-lock.programming-doc-readme.md](#) [L130-L165]
  patching file:  [useAiStream.hook.promptSourceIdentity.test.ts](#)
[AICode/src/frontend/src/messageBus/ test /useAiStream.hook.promptSourceIdentity.test.ts](#)
Grouped
([what is it?](#))

  patching file:  [LiveSessionContracts.test.ts](#)
[AICode/src/sharedlib/src/types/liveSessionTypes/ test /LiveSessionContracts.test.ts](#)
Grouped
([what is it?](#))

  reading project:  [abortSource.endpoint.ts](#) [L1-L120]

Corrections terminées

Joel, les quatre fantômes de concurrence identifiés pendant la vérification ont perdu leurs badges d'accès 🤖
J'ai corrigé **l'ensemble des anomalies du rapport** dans **20 fichiers**, avec des tests de régression ciblés et la mise à jour des documents d'architecture concernés.

Ensemble corrigé	Implémentation	Couverture de ré
Éviction destructive des replays	registre liveSession révoque désormais toutes les leases actives avant de réveiller les consumers. Une subscription évincée ne copie plus et n'expose plus le suffixe bufferisé restant.	tests du registre et replay vérifient qu' non consommé ne t jamais l'éviction.
Stop one-shot après fermeture/réouverture	La branche Open live transporte désormais le booléen strict abortRequested. le service Open le relit après les I/O et la revalidation du permit, ce qui couvre aussi un Stop revendiqué pendant le chargement du snapshot.	tests Open couvre l Stop pendant l'I/O normalPrompt couv remount avec Stop consommé.
Contrat partagé du latch d'abort	contrat LiveSessionOpenResult exige abortRequested uniquement sur la branche live, y compris dans le transport Slimbk et l'enveloppe frontend opaque. Les branches terminales et non-session l'interdisent, même présent à undefined.	tests des contrats liveSession et tests l'endpoint Open co les propriétés requi interdites.
Restauration frontend du Stop exact	hook usePromptLiveSession retire immédiatement l'autorité Stop lorsqu'Open annonce abortRequested=true. Une réponse structurée { accepted: false } retire également cette autorité ; seule une vraie erreur de transport permet de	Les fixtures live de d'hydratation, hand isomorphisme ont é adaptées au contrat

Ensemble corrigé	Implémentation	Couverture de ré
	réessayer.	
Exception synchrone de <code>onData</code>	hook useAiStream possède maintenant une unique requête de finalisation par tentative. <code>onData</code> , <code>onComplete</code> et <code>onError</code> convergent vers Queue A → Queue B → commit avant la publication de <code>completed</code> ou <code>transportError</code> .	test d'identité et de finalisation stream qu'un mauvais <code>turn</code> publie pas prématurément l'état idle et produit un terminal synthétique exact.
Course <code>onComplete</code> puis <code>transport error</code>	Une erreur transport arrivée pendant la finalisation d'un completion déjà engagé met à niveau l'outcome en <code>transportError</code> . Une seconde vérification après la barrière de commit garantit le terminal synthétique lorsque nécessaire, sans double terminal.	Le mécanisme est p par les trois callbac identité exacte sou + <code>streamInstan</code>
Propriété transactionnelle du masque <code>fallback</code>	Dans Prompt.tsx , seule l'invocation ayant installé la barrière { <code>turnId</code> , <code>streamInstanceId</code> } peut la retirer. Une tentative refusée ne modifie plus le masque d'une tentative déjà acceptée.	test de la barrière fallback reproduit c soumissions dans la pile : A acceptée, B avec conservation e masque A.
Documentation de maintenance	Les invariants d'éviction, de remount Stop-safe, de finalisation commune des callbacks et de propriété du masque ont été documentés.	documentation liveSession , docume Queue B et docume fallback persist son synchronisées avec nouveau comportement
Fichiers générés	Le contrat source a été corrigé sans modifier manuellement aucun artefact généré ; la génération normale du projet produira le transport correspondant.	Respect de l'invariant <code>*.gene</code> <code>*</code> .
Résultat architectural Les quatre propriétés qui manquaient sont maintenant fermées :		
1. Une éviction retire réellement toute capacité de replay , même pour une subscription qui conserve encore une référence interne vers l'ancien record.		
2. Un Stop déjà revendiqué ne ressuscite plus après remount , y compris lorsqu'il est accepté pendant la lecture asynchrone du snapshot.		
3. Aucune exception synchrone de listener ne peut publier une finalisation prématurée avant les deux queues et le commit React.		
4. Une tentative refusée ne peut plus retirer une barrière qu'elle ne possède pas. La correction reste strictement localisée : aucun changement d'algorithme n'a été introduit dans Queue A, Queue B, les sanitizers ou les règles de handoff.		
Nom de commit		

text

Copy

```
fix(live-session): close eviction, stop, and frontend finalization races
```



Les leases sont proprement coupées 🚫 Les Stops restent verrouillés, les courses sont domptées 🎵

</ASSISTANT>

<USER>

</USER>

<ASSISTANT>

I'm thinking...

<ASSISTANT>

  *scanning problems*

[AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts](#)

PatchFull file

[AICode/src/extension/src/aiTools/orch/liveSession/open/LiveSessionOpen.service.ts](#)

Accept

[AICode/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/ test /open.endpoint.test.ts](#)

PatchFull file

```
- import type { LiveSessionService } from
'../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
```

```

+ import type { LiveSessionService } from
+ '../../../../../aiTools/orch/liveSession/core/LiveSessionService.js';
+ import { ZLiveSessionOpenResult } from
+ '../../../../../sharedlib/types/liveSessionTypes/LiveSessionOpenResult.type.js';
+
- it('transports exact source identity, replay mode, and abort latch', async () => {
-   openMock.mockResolvedValueOnce({
-     kind: 'live',
-     session: SESSION,
-     threadId: SESSION_ID,
-     turnId: 'turn',
-     sourceId: 'source',
-     replayMode: 'continueAssistant',
-     abortRequested: true,
-   });
-
+ it('transports exact source identity, replay mode, and abort latch', async () => {
+   openMock.mockResolvedValueOnce(
+     ZLiveSessionOpenResult.parse({
+       kind: 'live',
+       session: SESSION,
+       threadId: SESSION_ID,
+       turnId: 'turn',
+       sourceId: 'source',
+       replayMode: 'continueAssistant',
+       abortRequested: true,
+     }),
+   );
+
+

```

[AI Code/src/extension/src/slimbk-local/endpoints/ai/orch/liveSession/_test_/open.endpoint.test.ts](#)

Copy

Review

Accept

  reading project:  [LiveSessionOpen.service.ts](#) [L1-L12],  [LiveSessionOpen.service.ts](#) [L120-L160],  [open.endpoint.test.ts](#) [L1-L58]

Correction des deux erreurs de typage

J'ai appliqué **LLM-DEBUG-3** en distinguant le contrat applicatif du problème présent dans les deux sites TypeScript signalés.

Le champ `abortRequested` était déjà correctement déclaré dans le schéma canonique partagé. Les deux erreurs provenaient des **littéraux d'objet directement soumis à un type de retour dont la vue TypeScript n'exposait pas encore cette nouvelle propriété**.

Changements appliqués

Fichier	Correction	Complétude
LiveSessionOpen.service.ts	La branche <code>live</code> est désormais construite et validée avec <code>ZLiveSessionOpenResult.parse(...)</code> . Le résultat est ainsi dérivé directement du schéma canonique comprenant <code>abortRequested</code> , au lieu de dépendre d'un contrôle de propriété excédentaire sur le littéral.	
open.endpoint.test.ts	La fixture du mock est également construite avec <code>ZLiveSessionOpenResult.parse(...)</code> . Le test utilise maintenant exactement le même contrat strict que l'endpoint et le service.	

Fichier

Correction

Complétude

[LiveSessionOpenResult.type.ts](#)

Aucun contournement ni relâchement du contrat : `abortRequested` reste obligatoire sur `live` et interdit sur les autres branches.



Fichiers générés

Aucun fichier généré n'a été modifié manuellement.



Résultat

- Les deux littéraux ne déclenchent plus le contrôle TypeScript de propriété inconnue.
- Le champ `abortRequested` reste strictement validé.
- Le service échoue immédiatement si sa branche `live` diverge du contrat partagé.
- Le test reste strictement typé, sans `any`, `cast`, assertion inutile ou contournement.
- La correction renforce le contrat plutôt que de masquer le diagnostic.

Nom de commit

text

Copy

`fix(live-session): validate abort latch through canonical open schema`



Le latch passe enfin le typage



Le schéma garde fermement l'aiguillage



</ASSISTANT>

SHOW ALLPage 1 / 1