Onderzoeksverslag: Mega-proactieve AI-bots voor Jamal's vloot

Vraag: hoe maak je Conductor, Sjakie, Coco, Paco en Jarvis proactief — verbanden leggen en voorspellen wat Jamal nodig heeft, vóórdat hij het vraagt.

Kort antwoord vooraf: er bestaat geen kant-en-klare "voorspel-Jamal"-technologie. Wel bestaan er bruikbare bouwstenen: een lichte controle-laag die per beurt beslist "stil blijven / kort ingrijpen / volledig helpen", plus geheugen-architecturen die zelf bepalen wat te onthouden. Niets hiervan vervangt het LLM — het zit er telkens naast of onder.

---

1. Algoritmes/ML-technieken: wat bestaat er, en wat is bruikbaar bovenop een LLM

Klassieke sequence-modellen bestaan al sinds 2010: FPMC combineert een Markov-keten (korte-termijn volgorde) met matrix-factorisatie (lange-termijn voorkeur) om te voorspellen welke actie iemand hierna doet.
Bron: https://cseweb.ucsd.edu/classes/fa17/cse291-b/reading/p811.pdf

Meta's HSTU-architectuur (2024) schaalt dit idee op tot triljoenen parameters en gaf in productie 12,4% betere metrics en tot 15x snelheidswinst. Dit bewijst dat sequence-modellen op actiegeschiedenis enorm kunnen schalen — maar Meta traint dit op miljarden gebruikers. Voor één gebruiker (Jamal) is dit qua datavolume niet haalbaar; dat is mijn eigen inschatting, geen conclusie uit het paper.
Bron: https://arxiv.org/abs/2402.17152

Het meest relevante recente werk is een controle-laag, geen voorspelmodel. PASK (april 2026) introduceert IntentFlow: een klein model dat per gespreksbeurt een token kiest — `<silent>`, `<fast_intervention>` of `<full_assistance>` — met 1,3-1,8 seconde latency (let op: níet milliseconden, dat was een fout in een eerdere versie van dit onderzoek). Dat is 2-3x sneller dan een los snel LLM zoals Gemini Flash of Claude Haiku, geen orde-grootte-sprong. Belangrijkste bevinding: veel modellen zijn goed in óf helpen óf zwijgen, niet in beide — gebalanceerde training op beide is cruciaal om niet opdringerig te worden.
Bron: https://arxiv.org/html/2604.08000v1

Zonder apart model, puur met het LLM zelf: ProAct (mei 2026) laat het LLM tijdens wachttijd tussen berichten alvast speculatieve vervolgvragen bedenken en retrieval voorbereiden — geen training nodig, alleen slimme prompting. Resultaat: 14,8% minder gespreksbeurten, 11,7% minder moeite voor de gebruiker, 28% minder hallucinaties. Dit is de goedkoopste, direct met Claude Code bouwbare vorm van proactiviteit.
Bron: https://arxiv.org/pdf/2605.25971

Een waarschuwing bij dit alles: TriggerBench (juni 2026) laat zien dat modellen die goed zijn in spontaan iets onthouden en erop acteren (prospective memory), ook meer valse alarmen geven. Modellen "overfitten" makkelijk op een altijd-waarschuwen-heuristiek, en nauwkeurigheid daalt sterk bij lange context of overlappende triggers — precies het risico bij een vloot met veel losse MEMORY.md-bestanden.
Bron: https://arxiv.org/pdf/2606.23459

Conclusie hoek 1: het bruikbare patroon is niet "een ML-model dat voorspelt wat Jamal wil", maar een lichte classificatie-laag (stil/kort/vol) plus idle-tijd-prompting van het LLM zelf. Beide zijn met Claude Code te bouwen, zonder los trainingstraject.

---

2. Praktijkvoorbeelden: wie doet dit al, en hoe

Er is geen publiek bewijs dat grote consumer-assistenten (Copilot, Gemini) een los ML-voorspelmodel gebruiken. Ze doen het via geheugen + retrieval, niet via sequence-modellen:

- **Microsoft 365 Copilot Memory** is een aparte laag van Recall: het onthoudt voorkeuren/instructies uit gesprekken, gekoppeld aan Microsoft Graph, en verrijkt nieuwe prompts daarmee vóórdat ze naar het LLM gaan. Puur retrieval, geen voorspelmodel.
  Bron: https://www.m365.fm/copilot-memory-vs-recall-shocking-differences-revealed/
  (kernclaim bevestigd, maar het aangehaalde voorbeeld in de brontekst zelf klopte niet — check bij gebruik Microsoft Learn direct)
- **Windows Recall** draait lokaal op de NPU, maakt screenshots en indexeert ze met een lokaal beeldmodel — puur retrieval voor latere zoekopdrachten, geen voorspelling. Let op: de beveiliging is minder waterdicht dan Microsoft claimt. In maart 2026 vond onderzoeker Alexander Hagenah (tool: TotalRecall Reloaded) een manier om versleutelde Recall-data alsnog te stelen via malware die al op het toestel draait. Microsoft noemt dit "by design", beveiligingsonderzoekers zijn het daar nadrukkelijk niet mee eens.

Bron: https://blogs.windows.com/windowsexperience/2024/06/07/update-on-the-recall-preview-feature-for-copilot-pcs/ (let op: dit is een vroege, voorlopige aankondiging — niet de definitieve 2025-versie)

- **Rewind/Limitless werd op 5 december 2025 door Meta overgenomen**, en per 19 december 2025 werden scherm- en audio-opname in de Mac-app uitgeschakeld — in sommige regio's (EU, VK, Brazilië, China) zelfs met volledige dataverwijdering. Dit is een harde les: "passief alles opnemen" als architectuur is kwetsbaar voor overname-risico en plotselinge privacy-ingrepen door een derde partij.
  Bronnen: https://9to5mac.com/2025/12/05/rewind-limitless-meta-acquisition/ , https://techcrunch.com/2025/12/05/meta-acquires-ai-device-startup-limitless/

De referentie-architectuur voor agent-geheugen komt uit het Stanford "Generative Agents"-paper (Smallville, 2023): memory stream (ruwe observaties) + retrieval-score (recency + importance + relevance) + reflection (hogere-orde inzichten) + planning. De retrieval-formule is simpel en direct bruikbaar: score = recency (exponentieel verval) + importance (LLM geeft zelf een 1-10-score bij aanmaak) + relevance (embedding-similarity). Geen apart getraind model nodig — puur LLM + een gewogen som.
Bronnen: https://ar5iv.labs.arxiv.org/html/2304.03442 en https://www.subodhjena.com/blog/generative-agents-memory-stanford

Let op: de claim dat dit patroon "de basis is van bijna alle latere systemen inclusief CrewAI" kon ik niet hard bevestigen in CrewAI's eigen documentatie — dat verband komt alleen voor in losse blogs. Gebruik het als invloedrijk precedent, niet als bewezen architectuur-overname.

Open-source geheugenlagen, direct bruikbaar bovenop Claude Code:
- **Mem0** is een bolt-on-laag: bij elke `add()`-call extraheert een LLM feiten en beslist zelf ADD/UPDATE/DELETE/NOOP om het geheugen consistent te houden. Cijfers uit het eigen paper: 26% betere kwaliteitsscore, 91% lagere latency, >90% tokenbesparing t.o.v. alles in de context proppen (self-reported door de makers, wel peer-reviewbaar via de methode zelf).
Bron: https://arxiv.org/abs/2504.19413
- **Letta (ex-MemGPT)** is een volledige agent-runtime met een OS-geïnspireerd geheugenmodel (core memory = RAM, archival = schijf) — de agent beheert zelf zijn geheugen via tool-calls.
Bron: https://www.letta.com/blog/agent-memory/

Voor Jamal's bestaande vloot (eigen agent-loop via Supabase-queue/Telegram) is een Mem0-achtige bolt-on-laag praktischer dan Letta overnemen, want dat laatste vervangt je hele runtime.

3. Data-bronnen en features: wat is het meest voorspellend

Uit de bovenstaande bronnen zijn dit de bruikbare signalen, in volgorde van praktische haalbaarheid met Jamal's huidige stack:

1. **Recency** — hoe recent is een taak/voorkeur genoemd. Simpel te implementeren als tijdstempel + exponentieel verval (Stanford-formule).
2. **Importance** — een LLM-toegekende score (1-10) op het moment dat iets wordt opgeslagen. Dit is al deels aanwezig in Jamal's systeem via de milestone-save-regel en team_rules.
3. **Relevance** — embedding-similarity tussen huidige context en opgeslagen feiten. Vereist een vector-store bovenop de Obsidian-vault/MEMORY.md-bestanden (nu ontbreekt dit — alles is platte tekst + grep).
4. **Herhaalpatronen/dagritme** — expliciet genoemd in de onderzoeksvraag, niet apart in de bronnen bewezen, maar logisch af te leiden uit Jamal's eigen daily-standup-ritueel (08:00) en de HEARTBEAT-structuur.
5. **Expliciete correcties/feedback** — dit is al deels de kern van Jamal's `feedback_*.md`-bestanden in MEMORY.md. Dat is feitelijk al een ruw geheugen-signaal-systeem, alleen niet gescoord of automatisch opgehaald.

Kritisch punt: geen van de gevonden bronnen beschrijft specifiek hoe je Obsidian-notities + chatgeschiedenis structureert tot voorspellende features. Dit is een gat in het onderzoek — je moet dit zelf ontwerpen, met de Stanford-scoring-formule als startpunt.

---

4. Risico's

- **Proactieve hulp voelt sneller bedreigend voor autonomie dan reactieve hulp.** Een Technion-studie (N=761 + N=571, voorgeregistreerd) vond dit significant. Belangrijke nuance: bij reactieve hulp voelde AI-hulp bedreigender dan hulp van een mens, maar bij proactieve hulp was er geen verschil tussen AI en mens. Het was wel hypothetisch vignette-onderzoek (geen echte AI-interactie), Amerikaanse deelnemers, nog niet peer-reviewed.
  Bron: https://arxiv.org/html/2509.09309v1

- **Valse alarmen/overfitting**: TriggerBench (zie hoek 1) toont aan dat modellen die goed proactief zijn, ook meer onterecht ingrijpen, en dat dit verergert bij lange context — precies Jamal's situatie met veel bots en veel geheugen.
- **Privacy/overname-risico**: het Rewind/Meta-voorbeeld (hoek 2) laat zien dat diepgaand gedrag vastleggen een derde-partij-risico is als je op externe tools leunt. Voor Jamal's eigen vloot (lokaal, eigen Supabase) is dit risico kleiner, maar wel iets om in het achterhoofd te houden bij elke nieuwe tool-integratie.
- **Balans stil/actief is de kern van het probleem**, niet een detail: PASK's bevinding dat modellen goed zijn in óf helpen óf zwijgen, niet in beide, is precies waarom "mega proactief" zonder rem al snel "irritant" wordt.

---

5. Concreet bouwplan-advies

Gegeven dat de dagelijkse proactieve briefing al bestaat, is dit de logische volgende stap — van eenvoudig naar geavanceerd:

1. **Voeg een expliciete importance-score toe aan elk MEMORY.md-item.** Bij elke `feedback_* / gotcha_* / reference_*` -toevoeging: laat de bot zelf een 1-10-score geven (Stanford-methode). Kost niets extra, alleen een prompt-instructie.
2. **Bouw een simpele recency+importance+relevance-retrieval bovenop de bestaande vault**, in plaats van alleen grep. Praktisch: één Mem0-achtige bolt-on-laag (of zelfgebouwde vector-index op de Obsidian-notities) die vóór elke sessie de top-N meest relevante items ophaalt — niet de hele MEMORY.md laden.
3. **Voeg een 3-staten-classificatie toe aan elke bot-beurt**, geïnspireerd op PASK: `stil / kort melden / volledig ingrijpen`. Dit hoeft geen los getraind model te zijn — een korte Claude-Haiku-achtige classificatie-prompt volstaat als eerste versie, gezien het kostenbewuste-model-principe uit Jamal's eigen regels.
4. **Experimenteer met idle-tijd-speculatie (ProAct-patroon) in Conductor**: tussen Telegram-berichten door, laat Conductor alvast 1-2 waarschijnlijke vervolgvragen bedenken en de bijbehorende info opzoeken — zonder dit te versturen, alleen klaarzetten. Pas versturen als het écht relevant blijkt.
5. **Test dit eerst op ÉÉN bot (Conductor), niet de hele vloot tegelijk.** Meet expliciet: hoe vaak is een proactieve suggestie raak versus overbodig/irritant (zelf-rapportage van Jamal,

bijvoorbeeld een simpele duim-omhoog/omlaag per suggestie in Telegram). Dat cijfer is je eigen precision/recall-signaal — gebruik het om de stil/actief-drempel bij te stellen, precies het probleem dat TriggerBench en PASK signaleren.

6. **Harde regel als vangnet, niet als los ML-filter**: net zoals enterprise next-best-action-systemen (Einstein/Pega) harde business-regels als filter gebruiken naast ML-scoring, moet elke proactieve suggestie altijd langs de bestaande KRITIEKE REGELS (geen cold outreach, geen haarvisie.nl, geen destructieve acties) — dit is een architectuurkeuze/analogie van mijzelf, geen bewezen enterprise-praktijk uit de bron, maar wel logisch en al deels aanwezig in Jamal's regelboek.

Wat niet gevonden/onzeker blijft: er is geen kant-en-klaar open-source project dat specifiek "Obsidian-vault + Telegram-bot-vloot + proactieve voorspelling" combineert. Alles hierboven is bouwstenen samenvoegen, geen bestaand template overnemen. Ook ontbreekt hard bewijs voor de optimale gewichten tussen recency/importance/relevance voor Jamal's specifieke situatie — dat moet empirisch getest worden, niet uit de literatuur overgenomen.