

AI mi vzala práci.

Pavel „snajpa“ Šnajdr

vpsFree.cz · vpsAdminOS

Moje výchozí prostředí

- **vpsFree / vpsAdminOS**

- Linux, OpenZFS, kontejnery; vývoj i provoz skutečné infrastruktury.
- Řešení závad napříč kernelem, filesystemem, kontejnery a konfigurací OS.

- **Moje dosavadní práce**

- Rozbor problému → návrh → implementace → ověření → nasazení.
- Znalost prostředí, souvislostí a důvodů jednotlivých rozhodnutí.

AI mi vzala práci.

- **První experimenty**

- oteam / cteam, dlouholeté potíže se ZFS, vlastní filesystem.
- Postupné zkoušení a velké ambice.

- **Další projekty a náklady**

- Sít', systémové nástroje a infrastruktura.
- Tokeny, předplatná, moje hypotézy o modelech a rozpracované projekty.

- **Praktická část**

- Moje pracovní postupy, nástroje a původní anglická zadání.
- Praktický příklad, otázky; celá příručka za slidy v PDF.

Experimenty s týmy: oteam / cteam

- **Prosinec 2025 až leden 2026 · éra GPT 5.1**
 - Delší běhy, různé role, pracovní větve a komunikace.
 - Pozorování jejich práce a způsobu domluvy.
- **oteam / cteam — experimentální nástroj**
 - tmux + klient, společný repozitář, samostatný výběr úkolů.
 - Krátký pokus s týmy agentů; repozitář snajpa/cteam.
- **První pozorování pracovního slovníku**
 - Requirements, scope, acceptance criteria, gates, decisions, state.
 - Podezřele časté opakování stejných slov v domluvě agentů.

github.com/snajpa/cteam · první commit 15. 12. 2025, další vývoj v lednu 2026.

ZFS: roky pádů a pořád stejná hláška

- **Potíže v produkci od roku 2021**
 - Opakovaný segfault userspace programu.
 - Pro mě celé roky diagnosticky nepoužitelná hláška; pátrání bez výsledku.
- **Poškozená paměť a její následky**
 - Souběh asi pěti až šesti podmínek, včetně reclaimu paměti na hostu.
 - Opakovaná hláška jako jediné průkazné vodítko.
- **Pokus v ChatGPT Pro 5.x**
 - Původní řádky logu vložené do obyčejného chatu.

GPT Pro: od logu k opravě OpenZFS

- **Řešení dlouholeté chyby od modelu → vlastní fork**
 - Jedna z oprav: `free_children()` a sdílené mapování bloků.
- **Rekurzivní průchod při souběžné změně dat**
 - Uvolnění zámku se zachovaným ukazatelem → ukazatel na neplatná data.
 - Držení zámku po celou rekurzi → opětovný pokus o zamknutí stejného mutexu, deadlock.
- **Kopie pod zámky → rekurze nad kopií**
 - Konzistentní pohled na data i po uvolnění zámků.
- **Celá série: 27 reportů → 24 patchů**
 - Po nasazení série konec popsanych dlouholetých pádů.
 - Upstream k 30. 9. 2026 stále bez této varianty `free_children()` s kopií dat.

[Oprava v našem forku](#) · [ověřená verze v upstreamu](#).

Od vlastního forku k vlastnímu filesystemu

- **Moje poučení ze ZFS**
 - Některé modely už teď schopnější než lidé.
 - ZFS: moje dosavadní příspěvky; Ceph: čerstvě nastudované vlastnosti a omezení.
- **VibeFS — vlastní filesystem**
 - Cíl: filesystem lepší než ZFS a Ceph.
 - Návrh architektury s modelem podle mých požadavků.
- **První prostředí: obyčejný ChatGPT chat**
 - Stovky iterací, Pythonový proof of concept, předávání projektu v tarballech.
 - Návrh, experimenty, testy i pracovní poznámky modelu.

VibeFS v chatu: stovky iterací přes tarbally

- **Tarball do chatu → práce modelu → nový tarball zpátky**
 - Zdrojáky, experimenty, testy a Markdown v jednom archivu.
 - Stovky turnů; každá další iterace s posledním stavem projektu.
- **Pythonový PoC připravený na převod do Rustu**
 - Explicitní stav, rozhraní a invarianty; co nejmechaničtější převod.
 - Výkonnější model pro návrh; levnější model pro pozdější přepis.
- **Dokumenty v tarballech z mého pohledu**
 - Především záznamy o právě dokončené práci modelu.
 - Poznámky k návrhu, experimentům a dalšímu kroku.

DeepSeek V4 Pro: 100 tisíc Kč a 1,3M řádků

- **Jméno VibeFS už obsazené → TideFS**
 - Pythonový návrh a původní pracovní dokumenty jako vstup.
 - DeepSeek V4 Pro; přepis a další vývoj v Rustu.
- **Přibližně 100 tisíc Kč za experiment přes API**
 - Zhruba 1,3 milionu řádků Rustu.
 - Velké množství vygenerovaných částí.
- **Výsledek přepisu**
 - Každý díl jako od nového autora při prvním setkání se zadáním.
 - Pokaždé jiná představa o užitečných vlastnostech výsledku.

Moje zaokrouhlené náklady; doložená útrata u DeepSeeku do konce srpna 2026: 4 423,48 USD, převážně za TideFS.

Řízení přepisu: moje „ifovačka“

- **Statický orchestrátor**
 - Ticket → krátké sezení agenta → PR → další ticket.
 - Koordinátor pro rozdělování práce a mergování PR.
- **Vstup každého workeru**
 - Aktuální kód, původní dokumenty z ChatGPT a přidělený ticket.
 - Pokaždé nová krátká relace s aktuálním stavem repozitáře.
- **Práce omezená na jeden ticket**
 - Vyřešený ticket, vytvořený PR, konec relace.
 - Další díl v rukou nového workeru.

Moje inspirace pro ifovačku

- **Pozorování z oteam / cteam**

- Vlastní nápad agentů OpenAI při delší práci: spolupráce přes tickety.
- Role a způsob domluvy podle aktuální potřeby projektu.

- **Moje následná volba**

- Tickety jako pevně daný způsob řízení práce na TideFS.
- Krátká sezení workerů; ztráta souvislostí a odpovědnosti za integraci celku.

- **Roztříštěný výsledek DeepSeeku → integrace modely OpenAI**

- Delší běhy modelů OpenAI, hledání rozporů a propojování jednotlivých částí.
- Víc času modelů oproti původnímu záměru; drahé poučení.

Podmínky dlouhé práce s modely OpenAI

- **Cíl, rozsah práce a oprávnění**
 - Požadované chování; dotčené části systému; povolené zásahy.
 - Nová oprávnění od člověka; běžná rozhodnutí v rámci převzaté práce od modelu.
- **Celý vývojový cyklus**
 - Requirements → specs → design → implementation → validation.
 - Zpětná vazba do návrhu i kódu; povolené dodání a případná správa infrastruktury.
- **Pracovní paměť mezi kroky a sezeními**
 - Aktuální stav, důvody rozhodnutí, pokusy, výsledky, zbývající práce.
 - Neúspěšný test → rozbor příčiny → oprava → pokračování v práci.

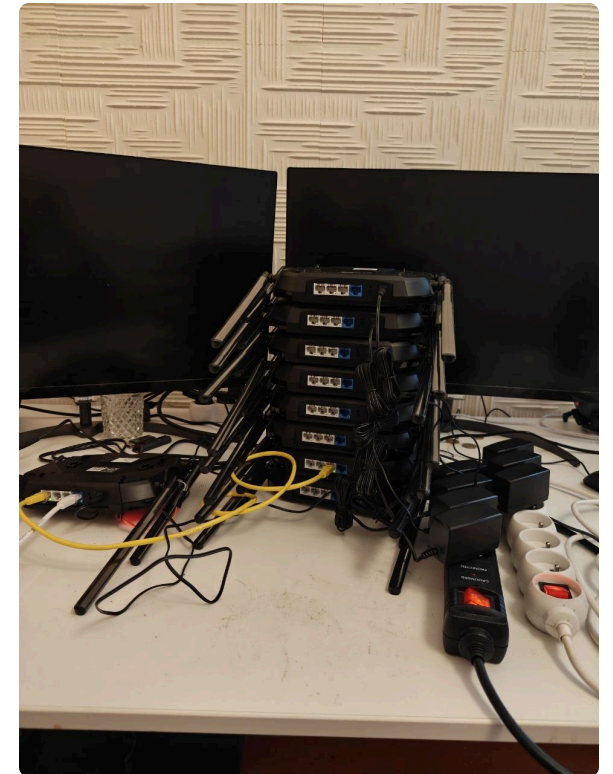
Secrety: moje důvěra v modely OpenAI

- **Důsledné odmítání čtení secretů samotnými modely**
 - Vlastní zdrženlivost modelu, bez mých opakovaných připomínek.
 - Stabilní chování při mé každodenní práci.
- **Moje zkušenost a důvěra z dlouhodobého používání**
 - Auditované rollouts v rozsahu stovek miliard tokenů.
- **Upozornění i při mém vlastním vložení hodnoty**
 - Po vložení secretu do chatu připomínka jeho výměny od modelu.

Moje dlouhodobá zkušenost s chováním modelů OpenAI; podrobnosti v podkladech.

MeshFleet: dvojice routerů a dostatek rádií

- **Mercusys MR85X · značka TP-Link**
 - Deset routerů celkem; 831 Kč za kus při původním nákupu.
 - Doma od 7. září 2026; vývoj kratší než měsíc.
- **Dvojice routerů na jednom místě**
 - Dostatek rádií pro frekvenční plán.
 - OpenWrt + BATMAN mesh; roaming, správa a měření propustnosti.
- **Jedna otevřená krabička s UARTem**
 - Přímý přístup podle instalačního návodu OpenWrt.
 - GPT 5.4 a exploit pro instalaci ostatních bez rozebírání.



vpsAdminOS: livepatche běžícího kernelu

- **Bezpečnostní backporty pro konkrétní běžící kernel**
 - Výběr relevantních oprav, přizpůsobení, ověření a sestavení kumulativního modulu.
 - Kompatibilita s konkrétním buildem, konfigurací a dotčenými funkcemi.
- **Příklad dodané kumulativní sady v5**
 - Výchozí kernel **6.12.95**; posouzené opravy z rozsahu **6.12.95** → **6.12.103**.
 - **47 bezpečnostních oprav** · **20 přiřazených CVE**.
 - Čísla pro tuto jednu dodanou kumulativní sadu.
- **Modely při údržbě infrastruktury**
 - Analýza dopadů, změny kódu, testy, podmínky integrace a provozní omezení.

Rozsah kumulativní v5 · výchozí kernel 6.12.95, hlášená verze 6.12.95.5.

Další dodané věci

codex-lb

Tracking fork; více předplatných, využití a limity.

snajpagent

Terminálový agent v produkčním nasazení.

aimux

Dlouhé úlohy v tmuxu, logy a oddělené projekty.

OpenZFS fork

Opravy pádů, backporty a vlastní systémové úpravy.

OpenWrt / MR85X

Vlastní firmware a instalace pro MeshFleet.

Vlastní Codex

Tracking fork; moje prostředí a provider adaptéry.

ibkrwgw

Gateway a dashboard; simulované obchody agentů.

eBPF exporter / RTNL

Měření čekání na rtnl_lock() a integrace do OS.

Kernelové livepatche

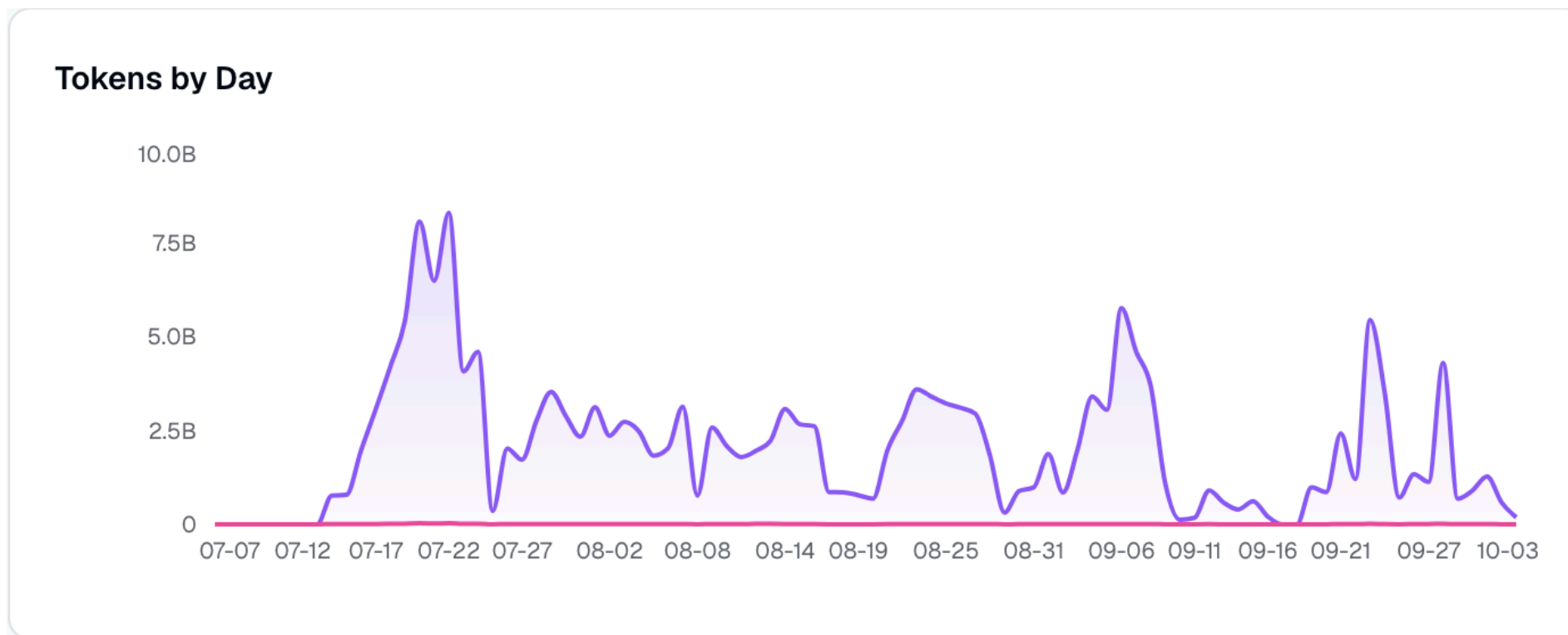
Posouzení oprav, backporty, buildy a dodané sady.

198,49 miliardy tokenů za necelé tři měsíce

Červenec **82,87 mld.**

Srpen **66,59 mld.**

Září do 28. 9. **49,03 mld.**



codex-lb · samostatný přehled k 3. 10. 2026

Součet a měsíce: denní statistiky čtyř Pro profilů, 1. 7.–28. 9. 2026, odečet 12:08 UTC; poslední den do odečtu.

Vstup včetně opakovaného kontextu a cache + výstup. Čtyři předplatná Pro; přesná čísla, náklady a metoda v podkladech.

codex-lb: účty, využití a dostupná kapacita

4 Pro + 2 Plus → codex-lb → Codex a snajpagent

Společné rozhraní a správa sezení, rozložení zátěže, limity a kredity

STATUS ↑↓	PLAN ↑↓	QUOTA ↑↓			SUBSCRIPTION ↑↓	PURCHASED ↑↓
● Active	Pro	Weekly	100%	🕒 in 6d 22h	50400.00	62500.00
● Active	Pro	Weekly	92%	🕒 in 6d 13h	46368.00	62500.00
● Active	Pro	Weekly	100%	🕒 in 6d 23h	50400.00	62500.00
● Active	Pro	Weekly	90%	🕒 in 6d 14h	45360.00	57278.75
● Active	Plus	5h	100%	🕒 in 4h 59m	7560.00	0.00
		Weekly	100%	🕒 in 6d 23h		
● Active	Plus	5h	100%	🕒 in 4h 59m	7560.00	0.00
		Weekly	100%	🕒 in 6d 23h		

Srpen 2026: 4 × přibližně 200 USD/měsíc za Pro
DeepSeek: 4 423,48 USD do konce srpna, převážně TideFS

Pro za 200 USD: poloviční kredity
Další experimenty podle dostupných tokenů

Výřez dashboardu ze 3. 10. 2026; sloupce Subscription a Purchased v kreditech.

Moje konspirační teorie: tick-tock

- **Menší model, vyšší úroveň schopností**
 - Střídání důrazu na efektivitu inference a zvyšování schopností.
 - Vyšší „floor“ i „ceiling“ při menším modelu — moje hypotéza.
- **Moje spekulace o počtu parametrů**
 - Největší model: lehce přes 1T; běžný asi 600B; menší asi 300B.
- **Cena inference a konkurence**
 - Zatím slabý tlak Anthropicu; malá motivace OpenAI ke zlevnění.
 - Další vývoj Anthropicu → očekávaný tlak na zlevnění OpenAI.
 - Moje zkušenost s Anthropicem: přetrvávající problémy s alignmentem.

Osobní hypotéza; T = bilion parametrů, B = miliarda parametrů.

Zvědavost: další možnosti AI

- **Vlastní zkoušení schopností modelů**

- Rozsah práce předatelné modelu; zbývající práce člověka.
- Cena dokončené práce a moje zkušenost s výsledkem.

- **Nahrazování lidí podle CEOs a moje zkušenost**

- Vývoj v rukou modelů → několik dní úplného volna → nuda → nové projekty.
- Nové možnosti a chuť do dalších experimentů.

- **Podněty pro další experiment**

- Další nápady na využití dostupné AI.
- Průběžné hledání schopnějších modelů a užitečných způsobů práce.

Aktivní: vpsAdminOS 6.18

- **Nové možnosti pro provoz s 6.18**
 - Tracing namespace: eBPF / perf a diagnostika vlastní VPS.
 - Proxy execution: běh vlastníka zámku za čekající úlohu; CPU účty vlastníka.
 - Proaktivní swap: první plánované nasazení s touto verzí.
- **Přenos a opravy vlastních funkcí z 6.12**
 - Virtuální swap a správa paměti kontejnerů.
 - Kontejnerové pohledy na CPU, paměť, /proc a /sys.
- **Stav k 2. říjnu 2026**
 - Linux + OpenZFS + LXC + integrační vrstva vpsAdminOS.
 - Integrační testy a opravy nálezů; kandidát před začleněním do stagingu.

Aktivní: vlastní LSM namespace uvnitř VPS

- **Funkční bezpečnostní mechanismy uvnitř kontejneru**
 - Ubuntu se zapnutým AppArmor.
 - Fedora s enforcing SELinuxem.
- **Vlastní LSM namespace při společném kernelu**
 - Vlastní policy ve VPS při zachování pravidel hostu.
- **Zbývající práce: kompletní policy pro vpsAdminOS**
 - Každá služba, její soubory, oprávnění a vazby na ostatní.
 - Dokončení a ověření hostitelské SELinux policy.

SELinux série: 54 patchů · 61 souborů · +9 491 / -1 902

Balík host policy: **1 675 řádků zdrojů · 27 souborů**, včetně sestavení.

Odečet 1. 10. 2026 · statistiky dokončovací SELinux větve; širší LSM historie zvláště v podkladech.

Aktivní: Tuzing — vlastní menší model

- **Porozumění pracovním dokumentům a zadání**
 - Význam, aktuální stav, autorita vstupů a rozhodnutí.
 - Pracovní paměť z modelů OpenAI jako podklad pro menší model.
- **Zlepšení v jednotlivých iteracích**
 - Dokumentové úlohy, stejná sada: **14/47** → **31/47**.
 - Správný další krok, samostatná sada: **13/31** → **25/31**.
- **Výrazný posun za malé peníze**
 - **Pod 200 USD celkem** podle mého dosavadního odhadu.
- **Další investice → naděje na použitelný model**
 - Celý úkol od zadání přes provedení a ověření po dodání.
 - Kvalitnější příklady, iterativní trénink a zachování získaných schopností.

Výsledky: dvě různé sady se srovnáním před/po k 1. 10. 2026. Náklady: můj odhad k 2. 10.

Součást vývoje: validace

- **Požadované chování → implementace → ověření výsledku**
 - Kompilace: syntaxe, typy, linkování. Funkční test: požadované chování.
 - Původní chyba, souběh okolností, integrace a výsledek při skutečném použití.
- **Neúspěch → zjištění příčiny → oprava → nové ověření**
 - Případně i oprava návrhu nebo chybného testu.
 - Oprava testu se zachováním požadavků na chování produktu.
- **Přiměřený rozsah ověření**
 - Cílený test konkrétní změny; podle potřeby zkouška celého systému.
 - Uložený výsledek náročného testu; další pokus zaměřený na nová zjištění.

Lidský vstup při ověřování

- **První společná prezentace**

- Můj způsob řeči; heslovité odrážky bez sloves; hustota slidů.
- Pořadí příběhu a moje představa o výsledku.

- **Moje připomínky → další práce modelu**

- Původní vstupy a korekce v pracovních poznámkách.
- Postupné zpřesňování porozumění mému stylu a záměru.

- **Fyzická pomoc u zařízení**

- Router: přepojení kabelu nebo podržení resetu.
- Průzkum, změny a další testování u modelu.

Dokumenty z TideFS

- **Dokumenty z tarballů, psané modelem během vývoje**
 - Požadavky, návrh, důvody rozhodnutí, pokusy a jejich výsledky.
 - Zachycená část mého uvažování o filesystemu i způsobu práce.
- **Rapid prototyping velkého, AI-native projektu**
 - Další přepisy podle získaného poznání; i možnost jiného jazyka.
 - Můj záměr po použitelném RC: přijímání AI příspěvků.
- **Z experimentu k obecnému pracovnímu notebooku**
 - Využití zkušenosti z filesystemu při práci na kernelu, infrastruktuře i dalších repozitářích.

Fieldbook: pracovní notebook a sdílená paměť

- **Markdown + Git, dostupné modelu v pracovním prostředí**
 - Zadání a jeho rozpracování; důvody rozhodnutí, aktuální stav, další krok.
 - Historie změn, jasně rozlišené návrhy a přijatá rozhodnutí.
- **Původní zadání vedle výkladu modelu**
 - Možnost návratu k přesnému původnímu vstupu člověka.
 - Postupné rozpracování do spec, návrhu, implementace a ověření.
- **Odpovědnost modelu i za pracovní záznam**
 - Čtení před další prací, porovnání se skutečným stavem, průběžná aktualizace.
 - Čerstvý agent bez historie session; stav i další krok z Fieldbooku.

github.com/snajpa/fieldbook · veřejný pracovní notebook v Markdownu a Gitu.

Předání úkolu od začátku do konce

- **Postupné předání částí vpsAdminOS a dalších repozitářů**
 - Kernel, ZFS, kontejnery, integrace OS; také konfigurace infrastruktury.
 - Od vlastní kontroly výsledků k předání psaní i čtení kódu modelům.
- **První testovací úkol od začátku do konce**
 - Návrh, změny, validace a nasazení v rozsahu udělených oprávnění.
 - Odpovědnost modelu i za navazující opravy a zápis skutečného stavu.
- **Moje souběžná práce**
 - Vlastní agent s IRC swarmingem: snajpagent.
 - Moje další nápady; průběžná péče GPT o předanou část vývoje.

snajpagent: vlastní terminálový agent

- **Malý agent v C, každodenní produkční používání**
 - Shell, soubory, nástroje a více poskytovatelů modelů.
- **Model: postup a dokončení práce**
 - Fieldbook: zadání, rozhodnutí a uchované souvislosti.
- **Runtime: běh a obsluha nástrojů**
 - Procesy, historie, výsledky a pokračování po turnu.
- **Přímý přístup k pracovnímu prostředí**
 - OS oprávnění procesu; izolace účtem, kontejnerem nebo VM.

snajpagent: soubory přes „devatero SSH hopů“

- **Přenos stejným terminálovým spojením**
 - Notebook ↔ SSH ↔ screen ↔ SSH ↔ vzdálený agent.
 - Soubory oběma směry; nativní wrapper `snajpagent remote`.
- **Vstup k modelu a výsledek zpátky ke mně**
 - `/receive` — místní soubor jako příloha vzdálené relace.
 - `/send CESTA` nebo `send_file modelu` — soubor do `~/Downloads`.
 - Ověření přenesených bajtů a potvrzení skutečné cílové cesty.
- **Běžný scrollbar a jednoduché ovládání**
 - Souvislé odpovědi; detaily nástrojů podle potřeby.
 - Méně stisků kláves v mém každodenním workflow.

Funkční přenos a testy přes SSH i GNU screen; technické podrobnosti v podkladech.

snajpagent: spolupráce a média přímo v agentu

- **Vestavěný IRC server i připojení do místnosti**
 - Lidé a modely na více strojích; společná domluva a předání práce.
 - Konverzace s agentem oddělená od společné místnosti; přímé oslovení agenta.
- **Dokumenty v textu i obrazu**
 - PDF, Office, tabulky, obrázky a výřezy.
 - Vybrané stránky nebo buňky přímo v nástrojích modelu.
- **Zvuk, video a hlas podle konfigurace**
 - Přepis a poslech audia, vzorky videa, generování řeči.
- **Provozní prostředí**
 - IRC v důvěryhodné síti nebo zabezpečeném tunelu.

Vestavěné nástroje a spolupráce; mediální API podle zvoleného poskytovatele a konfigurace.

Můj pracovní jazyk a slovník OpenAI

- **Moje běžná komunikace s modely: anglicky**
 - Moje zkušenost: silnější význam pojmů v celém anglickém zadání.
 - Samotné anglické výrazy v českých větách — slabší výsledek v mém použití.
- **Stejný slovník v dokumentaci OpenAI**
 - Acceptance criteria + validation commands; průběžná projektová paměť.
 - Progress, Decision Log, Validation and Acceptance; approval gates.
- **Pracovní pojmy s konkrétním významem**
 - Účel, podmínky a výstupy; vlastní názvy s výslovnou definicí.
 - Doslovné výskyty a významové souvislosti v samostatných podkladech.

[OpenAI: dlouhé úkoly](#) · [PLANS.md](#) · skutečná anglická zadání v následujících ukázkách; data zpráv v UTC.

Bootstrap prostředí → krátký běžný úkol

11. února 2026 · vpsAdminOS, Linux, ZFS a LXC · doslovný úryvek

```
take a look around and see what is vpsadminos, what components is it from, how  
to build it, and then figure out where kernel and zfs versions are tracked and  
where lxc version is tracked and map it onto branches and then try to figure  
out how I work with these manually, sum it up in the end and I'll tell you  
more/correct you
```

12. února · po seznámení s prostředím · celé zadání

```
yeah awesome so now please can you update vpsadminos 6.12 kernel? will you know  
what to do fully end to end? :)
```

Moje původní zprávy; úplný bootstrap a následný vznik ~/ai v příručce.

Celá vývojová smyčka

requirements → specs → design → implementation → validation

Část	Výstup pro další práci
requirements / specs	Původní potřeba a její přesné rozpracování.
design	Návrh řešení, rozhraní, rozhodnutí a jejich důvody.
implementation	Kód, konfigurace, integrace, testy a návody.
validation	Ověření požadovaného chování; nálezy a jejich příčiny.

- **Zpětná vazba:** nález → dotčené požadavky, specifikace, návrh nebo kód → nové ověření.
- **Dokumentace průběžně:** paměť práce i aktuální dokumentace produktu.
- **Dokončení:** ověřený celek a dodání v povoleném rozsahu.

Skutečný úkol: měření čekání na RTNL

- **Potřeba:** měření latence `rtnl_lock()` pro diagnostiku kernelu.
- **Výchozí projekt:** existující eBPF exporter a jeho příklad měření latence.

Moje celé zadání · 3. března 2026

```
clone https://github.com/vpsfreecz/ebpf_exporter to ~/tmp/ and do vpsadminos
branch there and in that branch, copy examples/accept-latency.bpf.c and modify
to rtnl_lock-latency.bpf.c, to measure latency of rtnl_lock() instead - hard
gate on actual validation that it's working, then commit and please push the
commit to the branch
```

Původní zadání do větve exporteru; pozdější integrace do OS jako následující úkol.

Goal, scope, ownership, authority

Pojem	Význam v prvním zadání exporteru
goal	Funkční měření latence <code>rtnl_lock()</code> .
scope	Exporter, nový příklad a jeho ověření; určená větev vpsadminos.
ownership	Zjištění podrobností, provedení, řešení chyb, ověření a dodání změny.
authority	Commit a push do určené větve po skutečné validaci.

- **Rozsah ze skutečného zadání a známého prostředí.**
- **Pracovní notebook:** úplné zadání, stav a další krok na filesystemu.
- **Trvalý runtime goal:** příležitost k dalšímu rozhodnutí po skončení turnu.

Navazující zadání: celá integrace do OS

3. března · moje kontrolní otázka

does it work?

Potom moje celé zadání integrace do vpsAdminOS

ok then please create a module in vpsadminos to be able to configure this exporter - look at how other exporters are done and inspire yourself by stealing that; then please add an integration test for it into vpsadminos, package this thing, etc. - integrate wholly please in one commit, push into staging when the test passes correctly (review that it does what it should afterwards please)

Stejná relace: původní měření → kontrolní otázka → nové zadání celkové integrace.

Requirements, specs, design a rozhodnutí

- **Requirements — potřeba z původního zadání**
 - Latence získání `rtnl_lock()`; skutečné ověření funkčnosti.
- **Specs — přesné chování podle projektu**
 - Sledovaná událost, výstupní metrika, jednotky, konfigurace a způsob ověření.
- **Proposal / design — možnosti a návrh zapojení**
 - Využití existujícího příkladu exportéru; vztah měření, konfigurace a služby.
 - Po rozšíření zadání také balíček, modul OS a integrační test.
- **Decision — zvolený postup a jeho důvod**
 - Rozlišení zvažované varianty a platného řešení; návazná implementace a validace.

Acceptance criteria a validation

- **Acceptance criteria — podmínky přijetí**
 - Měření skutečných událostí souvisejících s `rtnl_lock()`.
 - Po rozšíření zadání: konfigurovatelná služba a funkční integrace do OS.
- **Validation — zkouška těchto vlastností**
 - Spuštění exporteru s vybranou konfigurací.
 - Vyvolání práce se zámekem a kladný počet naměřených událostí v metrice.
- **Výsledek zpátky modelu**
 - Selhání → příčina → oprava implementace, návrhu nebo testu → nové ověření.
 - Podoba testu podle účelu; stále stejné požadované chování.

Výsledek integrace: služba a funkční test

- **Dodaná změna vpsAdminOS: 6 souborů, 170 přidaných řádků**
 - Balíček exporteru, konfigurační modul, registrace a integrační test.
- **Konfigurace v testovacím virtuálním stroji**

```
services.prometheus.exporters.ebpf = {  
  enable = true;  
  names = [ "rtnl_lock-latency" ];  
};
```

- **Test podle účelu měření**
 - Ověření aktivní konfigurace; přidávání a mazání rout.
 - Metrika ebpf_exporter_rtnl_lock_latency_seconds_count: podmínka kladného počtu vzorků.

[Commit integrace z 3. března 2026](#) · ukázka kódu testu z dodané změny.

Policy, gate a DONE

- **Policy — opakovaně platné pravidlo v daném rozsahu**
 - Projektové konvence, kompatibilita, oprávnění a ověřovací postupy.
- **Gate — podmínka dalšího kroku**
 - V původním zadání: skutečná validace před commitem a pushem.
 - V navazujícím zadání: úspěšný funkční test a kontrola jeho účelu před pushem do stagingu.
- **DONE — celý požadovaný a dodaný výsledek**
 - Implementace, integrace, ověření a dokumentace podle rozsahu úkolu.
 - Převzaté výsledky běžících testů; splněné podmínky dodání.

Výstupy v projektu a pracovním notebooku

Část	Repozitář produktu	Pracovní notebook
Requirements / specs	Rozhraní a projektové kontrakty.	Raw zadání, korekce, modelovo porozumění.
Design	Platný návrh a vztahy částí.	Varianty, rozhodnutí, důvody a odkazy.
Implementation	Kód, konfigurace, testy.	Rozdělaný stav a návaznosti.
Validation	Spustitelné zkoušky a postupy.	Revize, výsledek, log, zbývající ověření.
Documentation	Aktuální návod pro uživatele a vývojáře.	Stav a souvislosti další práce.

- **Průběžná aktualizace modelem; jedno platné místo pro každý význam.**

Gaps: porovnání se skutečným účelem

- **Proxy execution ve vpsAdminOS**
 - Cíl: omezení vzájemného blokování sousedních VPS na globálním zámku.

Moje celé zadání review · 8. června 2026

now, thoroughly review **everything** done for proxy exec thus far, identify gaps, where what we have is insufficient or not done

- **Implementation gap:** chybějící nebo nedostatečná část provedení.
- **Design gap / underdesign:** nedostatečný návrh pro skutečné použití.
- **Chybné porozumění:** řešení jiného problému.
- **Overimplementation:** práce navíc mimo skutečný účel.

Steering: moje oprava významu zadání

Navazující původní zprávy ze stejné relace · 8. června

```
i mean the issue which proxy exec needs to address is about global locks, one vps being a bad neighbor... if they do contention to themselves then thats not a problem and thus is not underimplemented - so pls identify this properly
```

```
yes and in that sense please also identify overimplemented places if we have any
```

- **Korekce:** sousední VPS na společném zámku; interní contention mimo tento cíl.
- **Dopad:** požadavky, spec, návrh, rozsah změn, testovací scénáře a pracovní paměť.
- **Ostatní zadání dál v platnosti; samostatné provedení opravy modelem.**

Paměť pro další krok a předání práce

Pojem	Vazba na další práci
current	Ověřený současný stav; po korekci také opravený význam úkolu.
work cursor	Zbývajících práce a nejbližší proveditelný krok podle current.
log	Historie pokusů a změn; důvod odlišného současného postupu.
evidence	Konkrétní revize, konfigurace a výstupy podpírající závěr.
handoff	Předání odpovědnosti s current, cursorem a potřebnými podklady.
blocker	Chybějící předpoklad konkrétního kroku; ostatní práce podle závislostí.

- **Po korekci proxy execution:** nové porozumění → kontrola dotčeného návrhu a testů → ověření účelu.
- **Před pokračováním:** porovnání zápisu se skutečným projektem a procesy.

Dva různé cykly zlepšování

- **Trénink modelu**
 - Hodnocení výstupů a změna vah; postupné zlepšování schopnosti.
- **Běžná práce s hotovým modelem**
 - Zlepšování produktu, návrhu, testů a pracovní paměti.
 - Další pokus podle výsledku předchozího; zachovaná zjištění a příčiny selhání.
- **Posouzení výsledku a vytvoření řešení**
 - Dvě různé schopnosti; funkční zkouška jako dostupná zpětná vazba.
 - RTNL: vyvolaná práce se zámkem → ověření vzorků v metrice.
- **Přínos paměti:** využití dosavadního poznání při dalším kroku i úkolu.

Moje kritika metodik a prompt engineeringu

- **Skills, OpenSpec a Anthropic-friendly směr kolem Claude / GLM**
 - Moje hodnocení velké části této reže: esoterika a ztráta času.
 - Organizace vstupu, nové informace a nástroje; schopnost modelu jako základ.
- **Moje zkušenost s modely OpenAI**
 - Celá práce v rukou schopného modelu; paměť a potřebné podklady.
 - Skills v tomto použití: především tokenová úspora a opakovaně dostupný obsah.
- **Moje výchozí cesta**
 - Pracovní notebook spravovaný modelem; minimum ruční obsluhy procesu.
 - Vlastní porovnání podle výsledku, tokenů, času a nutných zásahů.

Otázky?

- **Celá příručka za slidy v PDF**
 - Definice, původní anglická zadání, celý případ RTNL a vznik pracovní paměti.
 - Samostatné podklady za příručkou: odkazy, zdroje čísel a metodika.
- **Fieldbook** — github.com/snajpa/fieldbook
- **snajpagent** — github.com/snajpa/snajpagent
- **oteam / cteam** — github.com/snajpa/cteam

Jak předat modelu celou práci

Praktická příručka k přednášce **AI mi vzala práci**. — Pavel „snajpa“ Šnajdr

Vydání **2. října 2026**. Příručku najdete také za slidy v úplném PDF. Navazující [Podklady k přednášce](#) obsahují odkazy do dokumentace, zdroje čísel a další technické podrobnosti.

Modelům OpenAI už svěřuji všechnu práci na projektech, které jsem dříve vyvíjel. Ručně už kód nepíšu; k 2. říjnu je to zhruba tři čtvrtě roku od mého posledního řádku. Asi půl roku už jejich výstupy běžně ani nečtu. Dívám se na ně ze zvědavosti u zvlášť složitých úkolů nebo ve volné chvíli. Po předání vývoje jsem měl několik dní opravdu úplné volno. Pak mě to přestalo bavit a začal jsem vymýšlet další projekty. Dnes jsem zaměstnanější než před AI: pořád mě napadají nové věci a vybírám si doslova podle toho, na co mám tokeny.

Ukážu vám, jak modelu předat **celý vývojový úkol**, aby na něm samostatně pracoval až do dokončení. Na skutečných zadáních vysvětlím, co znamenají *requirements*, *design*, *gate*, *work cursor* a další pracovní pojmy. Používám pro ně také označení „OpenAI RL keywordy“. Postupně se dostaneme od zadání přes návrh a ověření až k pracovní paměti, díky které model při dalším úkolu využije to, co už zjistil.

Proč s modely pracuji anglicky

S modely se po celou dobu bavím anglicky. Pojmy jako *requirements*, *scope* a *gate* používám v celém anglickém zadání. Podle mé zkušenosti tak fungují lépe než jednotlivé anglické termíny vložené do českých vět.

U prvních GPT 5 jsem výslovně předával rozsah návrhu, implementace, testování i oprávnění k dodání. Novější modely snesou volnější zadání a zavedené pracovní podmínky čerpají z paměti. Myslím, že přesnými anglickými výrazy mířím na známé pojmy z úloh, na kterých se model učil. Příklady jejich použití v oficiální dokumentaci OpenAI najdete v úvodní části [podkladů](#).

Anglické ukázky jsou **moje původní zprávy**, včetně překlepů. U každé vysvětluji, co jsem potřeboval a jakou práci tím model převzal.

Co už model zná ze zavedeného projektu

V zavedeném projektu model najde pracovní postupy, potřebné souvislosti a platná oprávnění v instrukcích a pracovním notebooku. Do jeho odpovědnosti patří průzkum projektu, návrh, testování, řešení chyb i zaznamenání upřesněného zadání. Potřebná rozhraní, rozhodnutí a testovací příkazy si vyhledá a porovná s aktuálním stavem projektu.

Rozlišujeme tři situace:

- **První převzetí projektu:** model potřebuje najít projekt a notebook, poznat prostředí a zjistit rozsah oprávnění. Dostupné informace získává sám; člověk doplní podstatné nejasnosti.

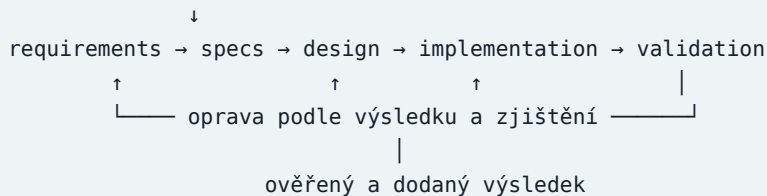
- **Běžný úkol:** člověk sdělí novou potřebu. Model ji zasadí do již známého prostředí a sám zajistí celý postup.
- **Korekce nebo výjimka:** člověk změní konkrétní význam či hranici. Model odvodí dopady, opraví porozumění a zachová nedotčené požadavky. Skutečná korekce proxy execution je v kapitole 8.

Krátký prompt se opírá o **dříve získané a průběžně udržované porozumění**. Nové zadání proto může být stručné: model už zná projekt, způsob práce i podmínky dokončení. Člověk přidá novou potřebu nebo opraví to, čemu model porozuměl špatně.

1. Uzavřený autonomní loop

Člověk popíše potřebný výsledek. Model vychází z platných oprávnění; případné změny jejich rozsahu doplní člověk. Model prozkoumá projekt, rozpracuje požadavky, navrhne řešení, provede změny a ověří je. Výsledek testu se vrací **modelu**, který podle něj opraví kód, návrh nebo pracovní postup a pokračuje. Průběžně udržuje pracovní paměť. Končí až po dokončení celého úkolu a dodání výsledku.

Záměr a oprávnění od člověka



Dokumentace vzniká průběžně: pracovní paměť zachycuje záměr, rozhodnutí a stav; produktová dokumentace popisuje dodané chování. Pokud test odhalí chybu v návrhu, model návrh opraví a promítne změnu do kódu i testů. Když zjistím, že špatně pochopil zadání, upřesním mu význam a zbytek opravy zajistí sám. O dalším technickém kroku rozhoduje podle výsledků a cíle úkolu.

K tomu model potřebuje dvě věci: klienta, který mu poskytne nástroje a umožní pokračovat v práci, a pracovní notebook, ze kterého načte souvislosti i po výměně modelu nebo začátku nové relace.

Konkrétní úkol pro následující výklad

vpsAdminOS je náš systém pro kontejnerové VPS. Potřeboval jsem diagnostikovat čekání na kernelový zámek `rtnl_lock()`, mimo jiné při práci se síťovými routami. Existující `ebpf_exporter` měl dodat toto měření jako metriku pro Prometheus.

3. března 2026 jsem nejprve zadal přípravu měření v repozitáři exporteru:

```
clone https://github.com/vpsfreecz/ebpf_exporter to ~/tmp/ and do vpsadminos branch there and in
that branch, copy examples/accept-latency.bpf.c and modify to rtnl_lock-latency.bpf.c, to
measure latency of rtnl_lock() instead - hard gate on actual validation that it's working, then
commit and please push the commit to the branch
```

Ve stejné relaci jsem později přidal balíček, konfigurační modul a integrační test ve vpsAdminOS. Úkol se tím rozšířil z exporteru na jeho zapojení do celého systému. Obě zadání a výslednou změnu ukazuje kapitola 12. Na tomto příkladu postupně vysvětlím pracovní pojmy.

Dva různé způsoby zlepšování

Při tréninku může hodnocení výsledků měnit váhy modelu. Při naší běžné práci se zlepšuje produkt, návrh, testy a pracovní paměť za použití stávajícího modelu. Další pokus se opírá o dosavadní stav **celého pracovního systému**. Uchování příčiny selhání umožní příště zvolit lepší postup a navázat na předchozí zjištění.

Posoudit výsledek a vytvořit řešení jsou dvě různé dovednosti. U exporteru lze i s omezenou znalostí implementace vyvolat událost a ověřit, že ji zaznamenal. Důležité je zvolit odpovídající zkoušku: kontrola startu procesu prověří spuštění služby, funkční test navíc ukáže, zda služba měří události. Model proto posuzuje, zda jeho zkouška skutečně odpovídá požadavku.

Krátké zadání ve známém prostředí

Při předávání vpsAdminOS jsem nejprve zpřístupnil prostředí, upřesňoval workflow a zadal jeho dokumentaci. Po předání tohoto kontextu jsem **12. února 2026** zadal:

```
yeah awesome so now please can you update vpsadminos 6.12 kernel? will you know what to do fully  
end to end? :)
```

První průzkum, moje korekce a vznik pracovních dokumentů jsou podrobně v kapitole 9. Krátké zadání se opírá o již získané porozumění. Model má sám zjistit aktuální stav, provést potřebné změny, ověřit výsledek a zachovat znalost pracovního postupu.

Cílové prostředí a oprávnění k dodání čerpá z platného zadání a pravidel. Fieldbook, kterému se věnuje kapitola 14, uchovává na filesystému celé zadání, rozhodnutí, aktuální stav i další krok. Trvalý runtime goal pak zajišťuje, aby model po dokončení jednoho turnu znovu dostal prostor pokračovat. Jeho text může být stručný: podrobnosti už jsou v pracovní paměti. Pokračování přes další turny zapnete výslovně v klientovi; ve snajpagentu slouží příkaz `/goal`.

2. Requirements, spec, goal a scope

Requirements — co má výsledek splňovat

Kdy: máte potřebu a chcete modelu předat i její rozpracování.

První požadavek u exporteru byl měřit latenci `rtntl_lock()` a skutečně ověřit funkčnost. V navazujícím zadání jsem přidal celé zapojení do vpsAdminOS. Rozpracování obou požadavků bylo součástí práce modelu.

U integračního zadání model zjistí existující způsob konfigurace exporterů v projektu a podle něj připraví balíček, službu a funkční test. Potřebné návody mají odpovídat skutečnému výsledku.

Co má vzniknout: přesný popis požadovaného chování, podmínky kompatibility a způsob ověření. Model přitom využije kód i dokumentaci projektu.

Spec — rozpracované zadání

Kdy: stručné zadání potřebuje doplnit technické podrobnosti, podle kterých lze dál pracovat.

Model z původního vstupu a průzkumu projektu vytváří pracovní spec. Rozlišuje požadované chování, omezení a vlastní rozhodnutí o návrhu. Originální vstup zachovává; vlastní rozpracování průběžně mění podle dalších zjištění a korekcí.

Co má vzniknout: modelovo srozumitelné rozpracování vašeho záměru. Nový model jej použije, aniž by musel znovu rekonstruovat celou konverzaci. Při pochybnosti se vrátí také k původním slovům.

Typická chyba: model při shrnutí změnil problém. Například místo vlivu dvou sousedních VPS řeší dvě vlákna jednoho kontejneru. Oprava pak musí zasáhnout spec, design i testy.

Goal — celý výsledek

Kdy: model přebírá odpovědnost za dokončení dlouhého úkolu.

Po rozšíření zadání je u exporteru cílem funkční měření ve vpsAdminOS. Model si podle toho vede přehled zbývajících práce a volí další krok: doplnění konfigurace, funkční test nebo dodání výsledku podle udělených oprávnění. Takto popsany cíl uchová v notebooku a řídí se jím při dalším rozhodování.

Co má vzniknout: dokončený výsledek v dohodnutém rozsahu, včetně zapojení do skutečného projektu.

Scope — co úkol zahrnuje

Scope prvního zadání končí měřením a jeho ověřením ve větvi exporteru. Navazující zadání výslovně zahrnuje balíček, modul, integrační test a dodání do vpsAdminOS. Potřebné změny návodů se řídí tímto rozsahem. Okolní monitorovací služby zůstávají zachované. Nesouvisející zjištění model uloží zvlášť a pokračuje ke svěřenému výsledku. Podstatnou změnu rozsahu podle potřeby upřesní člověk.

Scope pomáhá určit, které integrační změny úkol vyžaduje a kam až sahá. Funkční změna může potřebovat úpravy několika adresářů či repozitářů; její rozsah vychází z výsledku a souvislostí projektu.

3. Ownership a oprávnění

Ownership znamená odpovědnost dokončit svěřenou práci. Model si zjišťuje podrobnosti, rozhoduje o technickém postupu, řeší příčiny neúspěchů a dokončuje integraci. Sám zajišťuje návaznost jednotlivých fází.

Je to **trvalý způsob práce** zakotvený v projektu. Model odevzdává zapojenou, ověřenou změnu s odpovídající dokumentací. Rozhoduje podle pravidel projektu a výsledků ověření.

Authority určuje, o čem smí model rozhodovat a jaké zásahy má povolené. Ownership a authority se doplňují: model může odpovídat za celou opravu, která v daném zadání končí místními změnami a commitem. Při prvním převzetí projektu je potřeba vyjasnit, ve kterých repozitářích smí pracovat, jaké zásahy má povolené a kam a za jakých podmínek smí nasazovat. Při další práci model použije aktuálně platná oprávnění ze zadání a pravidel.

V prvním zadání exporteru jsem povolil commit a push do větve `vpsadminos` po ověření funkčnosti. Druhým zadáním jsem přidal dodání integrace do stagingu OS, opět po úspěšném funkčním testu. Z každého zadání tak model věděl, kam má výsledek dodat a za jaké podmínky.

Přístup nástrojů k prostředí určují oprávnění účtu a jeho nastavení. Přihlašovací údaje držte v přihlášeném klientovi nebo určeném chráněném úložišti. Technickou izolaci zajistíte vhodným účtem, kontejnerem či virtuálním strojem. Takto lze oprávnění a prostředí přizpůsobit konkrétní práci.

4. Design, proposal a decision

Design — jak a proč

Kdy: je potřeba vymyslet řešení odpovídající požadavkům.

Model navrhuje řešení podle requirements a stávající architektury. Uchová důvody klíčových rozhodnutí, způsob zapojení i očekávaná selhání. Běžná rozhodnutí o návrhu dělá sám a pokračuje implementací a ověřením.

U exporteru design propojí balíček, konfigurační modul, spuštění služby a testovací prostředí. U TideFS, mého experimentu s vlastním filesystemem, musí řešit i velikost metadat, trvalost potvrzených zápisů nebo přesun bloků používaných snapshoty.

Co má vzniknout: návrh, podle kterého lze řešení implementovat, a vysvětlení důležitých rozhodnutí. Další model tak pozná, proč řešení vypadá právě takto a co by si vyžádalo jeho změnu.

Proposal versus decision

- `proposal` je možnost, kterou ještě zvažujeme;
- `decision` je skutečně zvolený postup s důvodem a rozsahem.

Při porovnávání možností model rozliší zvažované varianty a zvolený postup. K rozhodnutí zapíše důvod a pokračuje implementací. Pokud nová zjištění ukážou chybu v původním rozhodnutí, upraví ho i navazující práci. Opuštěné varianty zachová jako záznam předchozích úvah.

Další model tak rozpozná platné řešení, najde jeho důvody a může posoudit případnou změnu.

5. Implementation a validation

Implementation je skutečná změna projektu včetně potřebného zapojení nové funkce.

Model doplní konfiguraci, registraci a testy potřebné pro použití. Vychází ze zvoleného designu a zavedených postupů projektu. Samostatně přechází od návrhu ke změně kódu a dalšímu ověření.

Validation ověřuje, zda výsledek splňuje požadavky. Způsob ověření navrhne a provede model.

Skutečný příklad: měření latence zámku

Model zapojil eBPF exporter do vpsAdminOS. Výsledná změna měla šest souborů a 170 přidaných řádků. Test obsahuje konfiguraci:

```
services.prometheus.exporters.ebpf = {
  enable = true;
  names = [ "rtnl_lock-latency" ];
};
```

Test ve virtuálním stroji přidává a maže routy. Tím vyvolá práci se zámekem a potom ověří, že metrika obsahuje kladný počet naměřených událostí.

Tento způsob ověření model odvodí z účelu měření. Vyvolá událost, ověří její záznam, zpracuje výsledek a případné chyby opraví. Po změně ověří také související části.

Typická chyba: model odevzdá „test selhal“ jako výsledek úkolu. Selhání testu pro něj obvykle znamená další práci: přečíst log, zjistit příčinu, změnit řešení a znovu je vyzkoušet.

Další typická chyba: test projde, ale ověřuje jiný problém. Při ověřování účtování CPU dvou sousedních VPS musí test skutečně použít dvě VPS s vlastními limity. Model proto posuzuje i vhodnost testu.

6. Acceptance criteria, policy, gate a DONE

Tyto pojmy se často používají vedle sebe. Každý má jinou funkci.

Pojem	Význam	Příklad pro exporter
acceptance criteria	Ověřitelné podmínky, které má výsledek splnit.	Zapnuté měření zaznamenaná událost vyvolanou ve VM.

Pojem	Význam	Příklad pro exporter
policy	Opakovaně platné pravidlo v určeném rozsahu.	Nové exportery se zapojují zavedeným způsobem konfigurace služeb.
gate	Nutná podmínka před dalším krokem.	Před publikací musí projít funkční test dokončené změny.
DONE	Celý svěřený úkol je dokončený.	Funkce, konfigurace, test, návod a povolené dodání jsou hotové.

Co model odvodí nebo převezme z projektu

Acceptance criteria model dopracuje podle skutečného použití. Policy převezme z pravidel projektu. U exporteru gate vyžaduje úspěšný funkční test: vyvolaná událost se musí objevit v metrice. Aby byl celý úkol DONE, model také aktualizuje návod a dodá změnu do určené větve. Podmínky dokončení si sám udržuje v poznámkách a řídí podle nich další práci.

Nově zaváděnou podmínku člověk sdělí. Model ji promítne do příslušného rozsahu práce a dál respektuje ostatní pravidla i skutečná oprávnění k publikaci.

Technickou podmínku, například úspěšný test, splní model sám. Pokud některý krok vyžaduje vaše rozhodnutí, uveďte které: například výběr cílového produkčního prostředí. Model pak ví, co má vyřešit samostatně a kdy se má obrátit na vás.

Kritéria mají vycházet z účelu výsledku. U exporteru je to zaznamenaná událost a funkční konfigurace služby. Zvolená zkouška má tyto vlastnosti skutečně prověřit.

7. Gaps: najít a odstranit rozdíly mezi zadáním a výsledkem

Kdy: model dodal mnoho práce, ale výsledek stále neodpovídá potřebě; nebo chcete před dokončením najít, co v řešení chybí.

Při práci na proxy execution ve vpsAdminOS jsem řešil, jak se sousední VPS navzájem zpomalují při čekání na společné globální zámky. VPS s vyčerpaným přidělem CPU může držet zámek, na který čekají ostatní. **8. června 2026** jsem zadal:

```
now, thoroughly review *everything* done for proxy exec thus far, identify gaps, where what we have is insufficient or not done
```

Model přečte uložené zadání a porovná requirements, spec, design, implementaci, validaci a dokumentaci se skutečným použitím. Rozliší níže uvedené příčiny, odstraní nalezené problémy i jejich důsledky v rámci zadání a ověří celek. Tuto kontrolu dělá také sám před dokončením. Krátký pokyn člověka ji může podle potřeby znovu zdůraznit.

Nález	Co se má změnit	Ověření opravy
Implementation gap	Doplnit zapojení hotového měření do služby.	Vyvolaná událost se objeví v metrice.
Design gap	Upravit datové struktury pro požadované měřítko.	Výpočet a zkouška paměťových nároků.
Chybné porozumění	Opravit spec a navazující řešení podle původní potřeby.	Porovnání s originálem a test správného scénáře.
Overimplementation	Zúžit řešení na svěřený účel.	Zachování potřebného chování po úpravě.

U filesystemu může být implementováno všechno plánované a přesto potřebujeme změnit návrh: metadata pro velký pool se nevejdou do paměti. I takovou chybu má model při samostatné práci odhalit.

8. Steering: upřesnění významu během práce

Na předchozí review proxy execution navázala moje korekce ve stejné relaci **8. června 2026**:

i mean the issue which prooxy exec needs to address is about global locks, one vps being a bad neighbor... if they do contention to themselves then thats not a problem and thus is not underimplemented - so pls identify this properly

Pak jsem výslovně doplnil také opačný směr kontroly:

yes and in that sense please also identify overimplemented places if we have any

Cíl se týkal sousedních VPS na globálních zámcích. Vnitřní soupeření úloh jedné VPS do tohoto konkrétního cíle nespadlo. Podle téhož účelu bylo potřeba hledat i zbytečně obecné části řešení. Následovalo celé zadání review:

ok so pls review what we have for this proxy exec thing so far in full according to goals, identify both underimplemented and overimplemented areas

Model má uchovat doslovnou korekci, opravit vlastní porozumění a určit dopady na spec, design, implementaci i testy. Potom provede opravu a ověření celku. Ostatní požadavky zůstávají v platnosti. Při další relaci vychází z opraveného zadání a aktuálního stavu.

Steering může dodat preferenci, oprávnění, nový fakt nebo opravu chybného výkladu. Provedení dotčených technických změn a údržba paměti dál patří do převzaté práce. Stejný princip platí pro změnu postupu: model využije uložené logy, zjistí příčinu, zvolí účelný další pokus a ověří dotčený celek.

9. Shared working memory: co model průběžně uchovává

Pracovní notebook má uložené původní zadání i modelovo vlastní rozpracování. Model jej sám udržuje, před rozhodováním čte a při nových zjištěních opravuje.

Obsah	K čemu slouží
Původní vstup a korekce	Vrátit se k tomu, co člověk skutečně zadal.
Spec a design	Pokračovat podle podrobné specifikace a návrhu.
Decisions a jejich důvody	Vědět, proč bylo řešení zvoleno a kdy je změnit.
Výsledky a neúspěšné pokusy	Použít předchozí zjištění při další volbě postupu.
Current a work cursor	Zjistit aktuální stav a další krok.

Údržba notebooku je součástí převzaté práce. Model zachovává původní zadání a korekce doslova, vlastní spec a design vede zvlášť. Před rozhodováním vyhledá relevantní poznámky, ověří je podle projektu a nová zjištění do nich promítne. Člověk tak může v dalším promptu sdělit přímo novou potřebu.

U jednoho malého projektu může stačit jedna poznámka. U více projektů přibude rozcestník. Git uchová vývoj rozhodnutí. Kód, testy a návody pro uživatele zůstávají v repozitáři produktu; pracovní notebook přidává záměr a souvislosti.

Jak vznikla moje pracovní paměť

Při experimentu VibeFS/TideFS jsem si z vývojových tarballů odnášel především dokumenty. Model si v nich zachycoval moje požadavky, vlastní porozumění, návrh a důvody rozhodnutí. Právě toto poznání jsem mohl využít při dalším přepisů i v jiných projektech. Stejný způsob práce jsem pak postupně předal modelům při vývoji vpsAdminOS. Z něj vychází i veřejný Fieldbook.

11. února 2026 — celé původní zadání prvního průzkumu:

now, this is my main development station for developing vpsadminos, which lives in ~/os and its kernel, zfs and lxc components, living in ~/linux, ~/zfs and ~/lxc - take a look around and see what is vpsadminos, what components is it from, how to build it, and then figure out where kernel and zfs versions are tracked and where lxc version is tracked and map it onto branches and then try to figure out how I work with these manually, sum it up in the end and I'll tell you more/correct you

Model měl prozkoumat čtyři repozitáře, jejich role, verze, větve a moje ruční workflow. 12. února jsem upřesnil zdroj pracovních návyků a požadoval jejich zápis:

yeah note the export NIX_PATH also in history and context all around that will tell you a LOT about workflows - and document them *all* in AGENTS.md right here in this folder

Doplnil jsem i aliasy a způsob práce s rebasem. Teprve po tomto seznámení přišlo krátké zadání aktualizace kernelu uvedené v kapitole 1. Při aktualizaci jsem doplnil, jak má model výsledek ověřit:

```
kick of make qemu instead but this is going to take a LONG time, like 2hrs is possible... but do it so that you know how make qemu works because I'll want you to always validate that the os actually works (and running test suite is for even longer so that we will sort another time)
```

Později ve stejné relaci následoval požadavek na zachování tohoto postupu:

```
make sure you document that you should use that to validate results when doing something that requires it, as full test suite is expensive (AGENTS.md)
```

Pracovní dokumenty postupně dostaly vlastní místo. Původní zadání přesunu většiny instrukcí:

```
move bulk of the instructions to ~/ai/docs from AGENTS.md and also structure the docs properly so that it can document my workflows, my work, what I want automated, etc... we'll have many projects with many subprojects probably, this one is vpsadminos with os,linux,zfs,lxc
```

První commit tohoto pracovního prostoru z **12. února 2026** už obsahoval rozcestníky, mapu repozitářů a samostatné workflow pro větve, build, validaci a dlouhé procesy. V postupu ověřování byl zaznamenán `make qemu`, včetně potřeby bootu VM a základních kontrol. Časově náročná celá testovací sada měla jinou roli než běžná iterace.

Paměť tedy vznikala přímo při práci: model prozkoumal prostředí, já opravil jeho porozumění, on zapsal pracovní postup a při dalším úkolu ho použil. Při návratu porovná poznámky se skutečnými větvemi, revizemi a výsledky. Já dodávám novou potřebu a podstatná upřesnění; model zajišťuje technickou práci i její návaznost.

Kde jednotlivé výsledky udržovat

Část práce	Projekt	Pracovní notebook
Requirements a spec	Existující rozhraní a projektové kontrakty.	Původní potřeba, korekce a modelovo rozpracování.
Design	Návrh důležitý pro další vývoj podle zvyklostí projektu.	Zvažované možnosti, důvody a odkazy na platný návrh.
Implementation	Kód, konfigurace, sestavení a testy.	Rozdělaný stav a návaznosti mezi částmi nebo repozitáři.
Validation	Spustitelné testy a zavedené ověřovací postupy.	Testovaná revize, závěr, zbývajících ověření a odkaz na výstup.
Documentation	Návod pro uživatele a vývojáře produktu.	Souvislosti další práce, current a work cursor.

Model určí, kde je platná projektová specifikace a návrh, odkáže na ně a při změně opraví související zápisy. Velké logy patří do samostatně uložených výstupů; notebook obsahuje potřebný závěr a odkaz na konkrétní místo.

10. Current, work cursor, work log a evidence

Current — co platí teď

Model průběžně aktualizuje popis stavu: co funguje, co chybí a jak to ověřit. Vychází přitom ze skutečných souborů a výsledků. Po korekci proxy execution tedy current zachytí správný cíl — vliv sousedních VPS na globálních zámcích — a stav opravy. Starší chybný výklad zůstává v historii spolu s důvodem změny.

Work cursor — kde pokračovat

Work cursor uchovává podstatná zjištění, zbývající práci, odkazy na výsledky a běžící příkazy i nejbližší možný krok. Při návratu model porovná zápis s aktuálním stavem projektu a běžících příkazů a naváže na předchozí práci.

Po korekci zadání model do cursoru zapíše, kterou část návrhu, implementace nebo testů má upravit jako další. U běžícího testu si uloží odkaz na proces, aby mohl převzít jeho výsledek. Po dokončení zapíše závěr a další krok.

Work log — co se stalo a proč

U případu proxy execution patří do historie původní review i následná korekce významu. Log zachovává důvod změny postupu, current popisuje nyní platné porozumění a skutečný stav práce.

Evidence — o co se opírá závěr

K výsledku testu model uchová revizi, konfiguraci, výstup a stručný závěr. Při další opravě využije tyto podklady. Na velký log odkáže tak, aby další relace našla potřebné místo.

Výsledek testu patří tomu kódu a konfiguraci, které skutečně běžely. Pokud se změní kód, konfigurace nebo předpoklady testu, model provede odpovídající nové ověření.

11. Blocker, handoff a souběžná práce

Blocker — konkrétní překážka dalšího kroku

Kdy: další potřebný krok závisí na chybějící informaci, přístupu nebo fyzické součinnosti.

Model pojmenuje chybějící předpoklad i krok, který blokuje, a pokračuje ostatní proveditelnou prací. Selhání pokusu sám vyhodnotí, určí příčinu a zvolí další postup v rámci svých možností a oprávnění.

U mé domácí sítě MeshFleet může být takovou překážkou potřeba přepojit ethernet nebo podržet reset. Tento krok udělám já a model potom pokračuje v softwarové práci. Za její dokončení dál odpovídá on.

Handoff — předání odpovědnosti

Předávající model zajistí, aby příjemce našel cíl a scope, relevantní původní zadání, spec, design a důvody, aktuální změny, výsledky i zbývající krok. Vyjasní, kterou část druhý model přebírá a kdo odpovídá za začlenění výsledku do celku. Přístupné podklady umožní dalšímu modelu samostatně navázat.

Co má vzniknout: další model rozumí tomu, co přebírá, má dostupné podklady a ví, jak navázat.

Jeden vedoucí, rozdělené části

Pokud zadání dovoluje souběžnou práci více modelů, vedoucí model rozdělí samostatné části, předá společné zadání a podklady, převezme zjištění, začlení změny a ověří celek. Udržuje společný návrh. Dál odpovídá za celý výsledek.

Vedoucí model vymezí pracovní kopie a odpovědnost za jednotlivé části, aby si pracovníci navzájem nepřepisovali rozpracované změny. Společný návrh a závěrečnou integraci má dál na starosti on.

Proč mi nestačil obecný controller

Zkusil jsem Nexus pro filesystem a potom obecný Codex Nexus. U prvního přibývaly větve a práce workerů, ale integrace stála; jeden worker psal `fsck`, přestože jej návrh filesystemu neměl potřebovat. Další dočasný model se znovu orientoval a další pravidlo v controlleru přidalo další větev řízení. Oba pokusy jsem opustil. Lépe mi fungoval model, který na jednom projektu pracoval delší dobu.

Při vývoji nové věci je potřeba zvolit další krok podle toho, co jsme zjistili. Stejně „test selhal“ může znamenat chybu implementace, špatný test, změněné prostředí nebo nevyhovující návrh. Vedoucí model musí rozlišit, **co se má změnit a co dál platí**, a podle toho upravit postup.

Vedoucímu modelu proto předejte také oprávnění upravit technické rozdělení práce podle zjištění. Když se změní společný návrh, určí dotčené workery, předá jim opravu, převezme změněné výsledky a dokončí integraci. Má na starosti celý úkol včetně závěrečného ověření.

12. Celý skutečný případ: měření RTNL a navazující integrace

Tato epizoda proběhla **3. března 2026 v jedné relaci**. Potřeboval jsem měřit latenci kernelového `rtnl_lock()`. Existující eBPF exporter měl vhodný výchozí příklad. Nejprve šlo o přípravu a ověření měření v jeho vlastní větvi, potom o integraci do vpsAdminOS.

1. První zadání — měření v exporteru

```
clone https://github.com/vpsfreecz/ebpf_exporter to ~/tmp/ and do vpsadminos branch there and in that branch, copy examples/accept-latency.bpf.c and modify to rtnl_lock-latency.bpf.c, to measure latency of rtnl_lock() instead - hard gate on actual validation that it's working, then commit and please push the commit to the branch
```

Goal: skutečné měření latence zámku. **Scope:** existující exporter, nový příklad a větev vpsadminos. **Authority:** commit a push do této větve za uvedené podmínky. **Gate:** skutečná validace funkčnosti. Model dostává celý technický úkol včetně zjištění podrobností, implementace a ověření.

2. Moje kontrolní otázka

does it work?

Chtěl jsem vědět, zda měření funguje. Potom jsem ve stejné relaci zadal jeho zapojení do vpsAdminOS.

3. Nový rozsah — úplná integrace do vpsAdminOS

```
ok then please create a module in vpsadminos to be able to configure this exporter - look at how other exporters are done and inspire yourself by stealing that; then please add an integration test for it into vpsadminos, package this thing, etc. - integrate wholly please in one commit, push into staging when the test passes correctly (review that it does what it should afterwards please)
```

Teď přibyly konkrétní požadavky: konfigurační modul podle existujících exporterů, balíček, integrační test a zapojení do OS. Změnu měl model dodat do stagingu vpsAdminOS po úspěšném testu. Požadoval jsem také, aby zkontroloval, zda test skutečně ověřuje požadované chování.

Model má z tohoto zadání a průzkumu projektu rozpracovat **specs**, zvolit **design**, provést **implementation** a zajistit **validation**. Rozhodnutí o podobě modulu se musí projevit ve skutečné konfiguraci a testu. Při selhání se zjištění vrací do dotčené části návrhu nebo provedení. Potřebná dokumentace se průběžně upravuje podle výsledku.

4. Výsledná změna a podoba funkční zkoušky

Commit integrace 534a026eed0a z 3. března obsahuje **šest souborů a 170 přidaných řádků**. Přibyl balíček exporteru, konfigurační modul, jejich registrace a integrační test.

Test spouští virtuální stroj se zapnutým exporterem a konfigurací `rtnl_lock-latency`. Ověří aktivní konfiguraci, přidává a maže routy a poté vyžaduje kladný počet vzorků v metrice `ebpf_exporter_rtnl_lock_latency_seconds_count`. Kontroluje tedy měření vyvolané události.

Takový test odpovídá původní potřebě: vyvolá práci se zámekem a zjistí, zda ji exporter skutečně změřil. Model má na starosti také spuštění testu, vyhodnocení výsledku a případnou opravu.

5. Co z tohoto případu přenést do další práce

Původní potřeba, rozsah, oprávnění a gate jsou rozpoznatelné v prvním skutečném promptu. Navazující zpráva přidává novou potřebu a mění rozsah. Technické rozpracování, integraci i volbu odpovídajících zkoušky zajišťuje model.

U delší práce si model průběžně zapisuje původní zadání, vlastní porozumění, rozhodnutí, výsledky a zbývající kroky. Příští relace poznámky přečte, porovná je s projektem a pokračuje. Stejně podklady pak využije i při navazující změně.

13. Skills, prompt engineering a OpenSpec

Základem samostatné práce je pro mě schopný model, který zná projekt a sám spravuje pracovní paměť. U Skills, prompt engineeringu a OpenSpec posuzují hlavně to, kolik odpovědnosti dokáže převzít model a kolik obsluhy přibude člověku. V Anthropic-friendly proudu kolem Claude a GLM vidím zásadně špatný směr: další údržbu receptů a lidské přepínání jednotlivých fází. Metodiku, která tyto činnosti přidává člověku a vydává je za cestu k autonomii, považují za esoteriku a ztrátu času.

Jak říkám, „z hovna bič neupleteš“. Základ je schopnost modelu, srozumitelná potřeba a skutečné porozumění projektu. Mně funguje schopné GPT a dlouhodobé dokumenty spravované modelem: vznikají při práci, zachycují poznání a další úkol ho může rovnou využít. Doporučuji si právě tento způsob práce vyzkoušet na vlastním projektu a posoudit, kolik práce tím převeze model.

Užitečnost receptu poměřuji tím, zda modelu dodá potřebné informace, nástroje nebo přesnější postup. Externí text poskytuje grounding: oporu pro práci se stávajícím modelem. Dotrénování mění samotný model. S modely OpenAI mi funguje předání celého problému a notebook, který si model udržuje sám.

Podklady a nástroje mají konkrétní hodnotu. Skill může dodat dobrý skript a šetřit opakované čtení. OpenAI Skills také podporuje; v jeho podání je chápu jako užitečný způsob, jak šetřit tokeny. OpenSpec může organizovat měnící se požadavky a návrh. Podstatné je, zda se díky tomu dokončí práce s menší lidskou obsluhou.

Pro vlastní porovnání použijte stejný model, projekt, informace, nástroje a oprávnění. Užitečný skript dejte oběma variantám, aby měly stejné nástrojové možnosti. Sledujte dokončený výsledek, čas, spotřebu a svoje zásahy. Započtete také vytvoření a údržbu instrukcí. Přidejte navazující úkol: u pracovní paměti se právě tam projeví hodnota předchozích zjištění.

Pro spolehlivější porovnání použijte několik odlišných skutečných úkolů, zachovejte i neúspěchy a předem určete, podle čeho přijmete výsledek. Testovací případy rozšiřte podle skutečného použití. Oddělte cenu prvního seznámení s projektem od navazující práce, kde se má uložené poznání zúročit. Potom posuďte, kolik práce vám další struktura ušetřila.

14. Fieldbook: začít s pracovním notebookem

Fieldbook je **přenosný pracovní notebook**, který modely udržují napříč projekty a relacemi. Používá obyčejný Markdown a Git. Obsahuje vstupní AGENTS.md, rozcestník projektů, slovník a ukázky pracovních poznámek.

```
git clone https://github.com/snajpa/fieldbook.git central-ai-notes
cd central-ai-notes
```

Máte-li už svůj notebook, nechte model začlenit užitečné části a zachovat jeho pravidla i historii. Při prvním zprovoznění mu zpřístupněte notebook a projekt. Klient má načíst vstupní AGENTS.md; model podle něj najde další podklady. Potom mu sdělte konkrétní potřebu a případně umístění informací, které dosud nezná. Takto proběhl i můj první průzkum vpsAdminOS v kapitole 9.

Model převezme také běžnou technickou údržbu. Poznává existující pracovní postupy, zachová původní vstup, rozpracuje jej a pokračuje až k ověřenému výsledku v daném rozsahu.

Pravidla o vlastnictví celého úkolu a údržbě poznámek přicházejí ze vstupních instrukcí. Model najde nebo založí potřebnou poznámku. Původní vstup zachová vedle vlastního rozpracování, udržuje current, work cursor a důvody rozhodnutí. Při dalším úkolu si je přečte a porovná s aktuálním stavem projektu. Kód a produktové návody zůstávají v příslušném repozitáři.

Nový model potřebuje funkční přístup k projektu a stejnému vstupnímu rozcestníku. Sám vyhledá potřebné poznámky a zjistí aktuální stav. Při prvním předání může vlastními slovy ověřit, zda projekt pochopil. Pokud se ukáže omyl, opraví svůj popis i dotčené části práce. Při běžném návratu naváže na existující zápisy a aktuální projekt.

Mohu tak spustit úplně čerstvého agenta bez historie relace, ve které se na úkolu pracovalo. Z Fieldbooku si zjistí, kde práce skončila, co už je vyřešené a čím pokračovat. Může přitom používat stejný model nebo jiný. Potřebné souvislosti má v souborech, které průběžně udržuje předchozí agent.

15. Snajpagent: terminálový agent pro tento způsob práce

Snajpagent už používám v produkci. Vlastního klienta Codex i codex-lb udržuji jako tracking forky: přebírám do nich další změny z upstreamu a zachovávám své úpravy.

Snajpagent je můj terminálový agent postavený na autonomii modelu. Model rozhoduje o postupu a používá nástroje; agent se stará o běh příkazů, jejich výsledky a pokračování relace. Proti mému běžnému workflow s Codexem mi přidává zejména přenos souborů přes vzdálené terminálové spojení a přímou spolupráci lidí a modelů přes IRC.

Práce bez překreslování celého okna

Text přibývá do běžného scrollbaru terminálu. Ve výchozím zobrazení čtu zprávy o průběhu práce; podrobnosti volání nástrojů jsou skryté. Zapínám je podle potřeby: `/verbose 1` ukáže stručnou aktivitu, `/verbose 2` také náhledy vstupů a výsledků.

Živé relace a výměna modelu

Živou relaci na Linuxu lze odpojit a znovu připojit. V relaci mohu změnit model a zachovat hotové výsledky nástrojů i běžící příkazy. Další model převezme kontext a pracovní notebook a pokračuje.

V nativním klientovi na Linuxu s ANSI terminálem používám tyto příkazy:

- `/s d` odpojí terminál a ponechá práci běžet;
- `snajpagent --attach` nabídne živé relace, `--attach ID` připojí konkrétní;
- `/s a ID` přepne na jinou živou relaci;
- `/goal ...` aktivuje pokračování přes další turny; `/goal pause` a `/goal resume` je pozastaví a obnoví. Podrobné zadání a stav uchovává Fieldbook.

Pro pokračování práce použijte odpojení živé relace. Běžný exit ukončí běh agenta. Běžící příkazy a jejich hotové výsledky zůstávají součástí příslušné relace.

Při potřebné korekci odešlu během práce upřesnění Enterem. Tabem přidám další úkol do fronty. Steering používám také v Codexu; ve snajpagentu mi vyhovuje jednodušší ovládání s menším počtem stisků kláves. Trvalý goal umožňuje modelu pokračovat i po skončení turnu podle stavu uloženého ve Fieldbooku.

IRC spolupráce

Snajpagent má IRC klienta i server. V místnosti se mohou domlouvat lidé a více modelů. Vedoucí model rozdělí práci, převezme výsledky a spojí je. Každá relace má vlastní kontext; dlouhodobé souvislosti uchovává pracovní notebook. Modely si sdělení předávají přímo v místnosti.

IRC provozujte v důvěryhodné síti nebo zabezpečeném tunelu; autentizaci a šifrování zajišťuje okolní prostředí. Do místnosti patří informace určené všem jejím účastníkům. Modelům proto vymezte, co mohou při spolupráci sdílet.

Přenos souborů přes stejné terminálové spojení

Nativní wrapper snajpagentu přenáší soubory oběma směry přes stejné terminálové spojení, včetně vnořeného SSH a GNU screenu. Na notebooku například spustíte spojení k vlastnímu pracovnímu stroji:

```
snajpagent remote ssh -t muj-pracovni-stroj snajpagent
```

Na obou stranách použijte sestavení podporující nativní přenos. Příkazem `/receive` zadejte místní cestu; po ověření přenosu se soubor zobrazí mezi přílohami vzdálené relace. Hotový vzdálený soubor odešlete přes `/send` CESTA, nebo požádejte model o `send_file`; wrapper jej běžně uloží do `~/Downloads` a vypíše skutečnou cílovou cestu. Přenos ověřuje bajty a potvrzení dokončení. Při chybějícím připojení wrapperu může export zůstat ve frontě pro pozdější doručení.

Dokumenty, tabulky a média

Model může číst dokumenty jako text i prohlížet jejich stránky, pracovat s buňkami tabulek a prohlížet obrázky. Podle konfigurace má k dispozici také práci s audiem a vzorky videa. Tyto nástroje používá přímo v relaci, vedle práce se soubory a příkazy.

Přístup k prostředí

Snajpagent používá přímo **OS oprávnění účtu**. Pro svěřenou práci mu zpřístupněte potřebné projekty a nástroje. Izolaci prostředí zajistěte vhodným účtem, kontejnerem nebo virtuálním strojem.

Secrety: moje zkušenost s modely OpenAI

Modely OpenAI při mé práci samy od sebe důsledně odmítají načítat obsah secretů. Drží se toho bez mých opakovaných připomínek a v mém workflow se na toto chování mohu spolehnout. Když jim citlivou hodnotu sám vložím do kontextu, upozorní mě na její výměnu.

Přihlašovací údaje může použít samotný proces, který načte chráněný soubor nebo předá hodnotu cílovému programu přes pipe. Model pak dostane potřebný výsledek s citlivými údaji zamaskovanými nebo vynechanými. Ochrana musí proběhnout **před odesláním výsledku modelu**. Použijte vypnutý shell tracing, omezené logy a odpovídající oprávnění. Secret vložený přímo do promptu se stává součástí kontextu; při úniku jej vyměňte.

16. Jak si ověřit, že díky modelům zvládáte víc

Zadejte skutečný úkol a potom navazující změnu. Sledujte, co ještě musíte dělat osobně:

- Dopracoval model requirements a navrhl řešení, nebo jste to celé udělali za něj?
- Zapojil změnu do projektu a odvodil testy z účelu?
- Po neúspěšném testu zjistil příčinu a pokračoval opravou?
- Promítl korekci do všech dotčených částí a zachoval ostatní požadavky?
- Při návratu využil pracovní notebook a hotové výsledky?
- Stačila další nová potřeba, nebo jste museli znovu opisovat známý kontext a způsob práce?
- Při souběžné práci někdo převzal integraci a dokončil celek?

Takto zjistíte, jakou část práce model skutečně převzal a dokončil a kolik ještě zbývá na vás.

Rozpočet vlastního pokusu přizpůsobte prvnímu konkrétnímu úkolu. Cenu dokončené práce snižuje také využití zachovaného poznání při další změně.

Další projekt vám může ušetřit čas. Nebo — jako mně — otevře prostor vymyslet si jich ještě několik.

Užitečná schopnost doma pod vlastní kontrolou

Dlouhodobě mě zajímá možnost mít schopné otevřené modely doma, nad vlastními daty a nástroji. Očekávám, že během příštího roku a půl až dvou let budou jejich schopnosti a dostupnost pro takové použití dál růst. Uchované záměry, rozhodnutí a zkušenost z práce pak využiji i s dalším modelem.

Navazující samostatná část: [Podklady k přednášce — dokumentace, projekty, čísla a metodika](#). V úplném PDF pokračuje hned za příručkou.

Podklady k přednášce

Zdroje a technické podrobnosti k přednášce **AI mi vzala práci.** — Pavel „snajpa“ Šnajdr

Vydání **2. října 2026**. Praktický postup najdete v [příručce](#). Zde jsou odkazy do dokumentace a zdrojového kódu, technické podrobnosti a způsob výpočtu uvedených čísel. U měření je vždy uvedené období nebo revize, ke které patří.

Pracovní slovník v oficiální dokumentaci OpenAI

Pracovní pojmy z příručky se objevují také v dokumentaci OpenAI. Následující odkazy ukazují jejich použití při dlouhých úkolech, průběžném ověřování a udržování projektové paměti. Přehled odpovídá dokumentaci z **30. září 2026**.

Acceptance criteria, validation a dlouhá práce

[Run long horizon tasks with Codex — durable project memory](#) uvádí přímo **„Acceptance criteria + validation commands per milestone“**. Oddíl [Verification at every milestone](#) propojuje průběh práce s ověřováním a opravami po neúspěšném testu. To odpovídá našemu rozlišení: co má být pravda, jak to zjistím a co udělám po selhání. Článek zároveň pracuje s průběžně udržovanými projektovými dokumenty.

Progress, Decision Log a ověřitelné podmínky přijetí

[Using PLANS.md for multi-hour problem solving — vzor PLANS.md](#) obsahuje názvy **Progress**, **Decision Log** a **Validation and Acceptance**. Přijetí výsledku váže na chování, které lze vyzkoušet. Název ExecPlan článek sám zavádí a definuje pro tento postup. Podobně zde používáme vlastní stručný název `work cursor` pro záznam o tom, kde a jak pokračovat. Jeho význam je vysvětlený v kapitole 10 příručky.

Gates v různých pracovních postupech

[Iterating Development Workflows with Codex](#) používá **„expected approval gates“**, lidské schválení fáze a zpětný rozbor práce. Rozlišuje rozpoznávaný `AGENTS.md` od ostatních volitelných názvů a uspořádání souborů. Jde o konkrétní recept s popsáním rozdělením práce.

V přednášce je gate obecně podmínka pro další krok. Může jí být úspěšné dokončení existujícího testu. Lidské rozhodnutí patří tam, kde ho vyžadují pravidla oprávnění nebo povaha úkolu. Konkrétní způsob pokračování a strukturu dokumentů přizpůsobujeme projektu. Model uvnitř těchto podmínek řídí běžnou technickou práci a sám udržuje paměť.

Autonomy, persistence a dokončení

[Codex prompting guide](#) — [recommended starter prompt](#) propojuje autonomii s vytrváním u úkolu a dotažením práce; rozlišuje také dokončené, blokové a zrušené položky plánu. Rozsah práce i oprávnění zasahovat do okolního prostředí určují zadání a pravidla projektu.

Sloveso gate se zde objevuje také ve významu podmínění či označení záměrných změn. Význam approval gates dokládá přesněji předchozí odkaz na Iterating Development Workflows.

Instrukce, paměť a oprávnění

[AGENTS.md](#) popisuje instrukce v repozitáři; [Security](#) hranice klienta a oprávnění. Pro využití paměti musí daná relace potřebný soubor načíst. Technický rozsah nástrojů určuje prostředí a oprávnění. Pojmy scope, ownership, policy, evidence či handoff používáme ve významu běžném při vývoji softwaru; zde je vysvětluje příručka.

Podklady příběhu, čísel a technických výsledků

oteam / cteam: pozorování spolupráce

Krátký experiment z éry GPT 5.1 je uložený v [snajpa/cteam](#). Historie začíná commitem 8324555 z **15. prosince 2025** a pokračuje do **17. ledna 2026** (656cadf, data v místním čase +01:00). Repozitář zachycuje změny rolí a způsobu domluvy agentů. Nechal jsem modely spolupracovat v angličtině, kterou s nimi používám od začátku, a pozoroval, jak si rozdělují práci a zavádějí vlastní postupy založené na ticketech. Tehdy jsem si poprvé všiml, jak často se v jejich domluvě opakují konkrétní slova: requirements, scope, acceptance criteria, gates, decisions a state. Při vlastní práci jsem tyto pojmy začal používat až později.

OpenZFS: roky pátrání a následná oprava

Opakovaná hláška o userspace segfaultu pro mě byla celé roky diagnosticky nepoužitelná. Problém vyžadoval souběh asi pěti až šesti podmínek včetně reclaimu na hostu. Poškozená paměť způsobovala zavádějící následky a opakovaná hláška zůstávala jediným průkazným vstupem. Až GPT Pro 5.x z ní v chatu dokázalo odvodit řešení.

[Downstream commit db78ba1962f5](#) ukazuje jednu opravu v celé sérii: free_children pro rekurzivní průchod pořídí pod zámky úplnou kopii nepřímého bloku a následně čte tuto kopii. Uvolnění zámku nad původními daty ohrožovalo platnost ukazatele; držení zámku po celou rekurzi mohlo způsobit deadlock.

[Upstream soubor](#) v připnuté revizi master z 29. září 2026 stále pracuje s původními daty. Při přípravě celé série vzniklo **27 IR reportů, zpracovaných do 24 normalizovaných patchů**. Po nasazení série produkční pády ustaly.

VibeFS / TideFS → DeepSeek → integrace OpenAI

Do ZFS jsem už přispíval, čerstvě nastudovaný jsem měl Ceph. Na IRC mi řekli, ať AI patche neposílám; můj dojem byl, že si jejich vznik představují s mnohem slabšími modely. Po zkušenosti s opravou jsem se pustil do filesystemu lepšího než ZFS a Ceph. VibeFS vznikalo v ChatGPT chatu: stovky iterací, Pythonový proof of concept připravený na převod do Rustu, tarbally se zdrojovým kódem i modelovými pracovními poznámkami. Moje obsluha znamenala asi 300krát napsat „go“ a občas přesunout poslední tarball do dalšího okna, když se chat rozbil. Další turn nebo čistý chat dostal poslední uložený stav a potřebné podklady; samotný vývoj, testy a další postup řešil model.

Jméno VibeFS už bylo obsazené, takže následovalo přejmenování na TideFS. Pythonový návrh a původní dokumenty dostal **DeepSeek V4 Pro** pro přepis a další vývoj v Rustu. Experiment stál zhruba **100 tisíc Kč** a vzniklo asi **1,3 milionu řádků**. Výsledek byl roztříštěný: jednotlivé části působily jako práce stále nového autora s jinou představou o filesystemu.

Použil jsem statický ticketový orchestrátor, svoji ifovačku. Krátká relace odpracovala ticket a skončila PR, koordinátor vytvářel tickety a mergoval změny. Inspirovalo mě dřívější pozorování spolupráce agentů OpenAI, ale já z jejich vlastní domluvy udělal rigidní rozdělení do krátkých sezení. Ztrácely se souvislosti a odpovědnost za integraci celku.

Teprve tento roztříštěný výsledek po DeepSeeku musely **modely OpenAI** v delších bžích propojit a dotáhnout. Stálo to další čas modelů oproti původnímu plánu. Tato zkušenost mě přivedla k dlouhodobému předání odpovědnosti za celý úkol.

Nejcennější zůstalo uchované porozumění v dokumentech: moje požadavky, důvody rozhodnutí, invarianty i poučení z omylů. Díky tomu je dostupné rychlé prototypování velmi velkého projektu, další zásadní přepis i možnost jiného jazyka. TideFS zůstává AI-native experimentem; po použitelném RC chci přijímat také AI příspěvky. Dnešní veřejný Fieldbook nabízí obecnější přenositelnou podobu modely udržované pracovní paměti.

Livepatch a rozpracované výsledky

Přehled livepatchů 6.12.95 v připnuté revizi dokládá v5: výchozí kernel 6.12.95, hlášená verze 6.12.95.5, rozsah **v6.12.95..v6.12.103**, **47 bezpečnostních oprav / 20 přiřazených CVE**. Čísla se vztahují k této nasazené kumulativní generaci.

MeshFleet, předání práce a používané nástroje

MeshFleet je nasazená domácí síť s routery **Mercusys MR85X**; celkem jich mám **deset**. Mercusys patří do rodiny TP-Link; výrobce popisuje model a jeho rádiovou výbavu. Klíčové jsou dvojice routerů na jednom místě: dost rádií pro frekvenční plán. Síť používám od 7. září 2026. Při původním nákupu stál kus **831 Kč**. Vývoj před nasazením trval méně než měsíc.

Fotografie zachycuje moji vývojovou sestavu. Jednu krabičku jsem otevřel a připájel UART, aby se do ní model dostal přímo. Postupoval jsem podle instalační cesty v návodu OpenWrt. Všechny routery se mi rozebírat nechtělo, takže **GPT 5.4** vyřešilo instalaci přes exploit. U ostatních jsem přepojoval

kabel a model prováděl výměnu původního firmwaru za OpenWrt bez otevírání zařízení. Dál se stará o konfiguraci, roaming, měření a opravy. Instalační podklady: [stránka zařízení OpenWrt](#).

Vlastní Codex i codex-lb jsou tracking fork; snajpagent je u mě v produkčním nasazení. Modelům OpenAI už svěřuji všechno, co jsem dříve vyvíjel, a ručně už kód nepíšu. K 2. říjnu je to zhruba tři čtvrtě roku od posledního vlastního řádku a asi půl roku bez běžného čtení jejich výstupů. Dívám se na ně ze zvědavosti u zvlášť složitých zadání nebo ve volné chvíli. TideFS zůstává filesystemovým experimentem a zdrojem poznání i pracovní paměti.

Rozšířený přehled dodaných výsledků

Mezi používané nástroje a dodané části patří: codex-lb, vlastní Codex, snajpagent, Fieldbook, ibkrwg, aimux, eBPF exporter/RTNL, OpenZFS fork, kernelové livepatche a OpenWrt pro MR85X. U codex-lb/Codexu jde o udržované tracking fork. Podklady pro RTNL, ZFS, livepatch a síť jsou uvedené v příslušných částech tohoto dokumentu.

ibkrwg je Rust gateway a nasazený dashboard pro virtuální účty agentů, se simulovanými obchody nad tržními daty. Zdrojem je projektový README a revize 35d85a0 (Add IBKR API budget controls). **aimux** je používaný lokální nástroj pro dlouhé úlohy v tmuxu, logy a oddělené projekty; zdrojem je jeho implementace bin/aimux a provozní dokumentace. **OpenWrt/MR85X** označuje vlastní firmware a instalační část už používané sítě MeshFleet.

snajpagent — terminálový agent a konkrétní odlišnosti

Snajpagent je terminálový agent v C. Soubory přenáší nativní wrapper snajpagent remote, příkazy /receive a /send a nástroj modelu send_file. Nativní protokol je **trzsz-kompatibilní**. Na pracovní stanici běží wrapper, na vzdáleném stroji sestavení agenta s podporou nativního přenosu.

V repozitáři jsou také příklady zapojení v testech test_remote_ssh.py: jeden SSH hop, SSH se vzdáleným GNU screenem a dva vnořené SSH hopy. Kontrolují přenesené bajty, cílovou cestu a potvrzení převzetí. test_remote_terminal.py zahrnuje vnořený screen a obnovu rozepsaného vstupu.

Proti mému běžnému workflow s Codexem mi snajpagent přidává zejména **terminálový přenos souborů a IRC server i připojení pro lidi a modely na více strojích**. Jeho ovládání a další nástroje popisuje kapitola 15 příručky. Pro uživatele Codexu jsou užitečné také [oficiální přehled funkcí CLI](#) a [přehled příkazů](#).

Rozpracované projekty — stav k 2. říjnu 2026

vpsAdminOS 6.18 — porovnání s naším 6.12: zdrojové srovnání z 1. října vychází z kernelů připnutých v os/packages/linux/available-kernels.nix v OS revizi

1a2f0264a8fa71d8fdeaf581cdcc5b4662b0fb03: stabilní **6.12.95**,

a2384967b90f24d2470c9eb15f0e66d938df7e08, a kandidát **6.18.49**,

108c524f775e9a1745ef03d6059de4dccc33a047. Zahrnuje downstream série vůči příslušným upstream základům a konkrétní implementace:

- **Nově proti této 6.12:** tracing namespace, kontejnerové eBPF tokeny, omezení přístupu k sondám a kontejnerová hranice perf. Konkrétně `include/linux/tracing_namespace.h`, `kernel/tracing_namespace.c`, `kernel/bpf/token.c` a `kernel/events/core.c`. Základ namespace je v patchi `b600dd9adfa7c`, BPF substrát v `fb48c7289fc11`, perf omezení v `1c0edde367bd4`.
- **Nově proti této 6.12:** proxy execution a downstream rozšíření sledování vlastníků `mutex/rwsem/percpu-rwsem/rtmutex` a `procfs` zámku (`37606d7642c08`), účtování běžícímu vlastníku (`99d15461d92a7`) a diagnostika (`2f322afce6b9a`). V 6.12 chybí odpovídající `find_proxy_task` a donor infrastruktura v `kernel/sched/core.c`.
- **Novinka pro plánované nasazení s 6.18:** proaktivní swap, jehož první použití plánujeme právě s touto verzí. Mezi přenášené a ověřované funkce patří virtuální swap, paměťové/CPU pohledy kontejnerů a vlastní syslog namespace.

Integrace zahrnuje Linux, OpenZFS, LXC a OS. Před začleněním celého vydání do stagingu probíhají integrační testy a opravy nalezených chyb.

LSM namespace — AppArmor i SELinux: uvnitř VPS fungují Ubuntu s AppArmorem i Fedora s enforcing SELinuxem. Zbývá dokončit celou hostitelskou policy pro vpsAdminOS, pro každou jednotlivou službu a její vztahy k ostatním. Širší LSM implementaci lze studovat v linuxové revizi `a22fd8f542bbc606178d90e1a23261028fbc6649`, včetně AppArmor backendu `aa_lsmns_backend`.

Následující statistiky patří samostatné dokončovací sérii s pevným SELinux backendem. Výše uvedený kernelový pin 6.18 tuto LSM sérii ještě neobsahuje.

SELinux / vlastní LSM namespace: kernelová série měřená 1. října obsahuje **54 lineárních commitů**, tj. 54 patchů. Čistý rozdíl od základu `e1895b740782e3ac4e4db75c16c05989022d4409` po `3eeca413c4c4f38d750db41285c4e58671de84c`: **61 souborů, +9 491 / -1 902 řádků**. Počet řádků je výsledek `git diff --shortstat base head`, tedy čistý rozdíl celé downstream série.

Vlastní balík host policy `os/packages/selinux-policy-vpsadminos/` v OS revizi `c5a2e2e1ff4983d0d5f1a13e40128ed113bfa0d4` má **1 675 fyzických řádků v 27 sledovaných souborech**, včetně komentářů a prázdných řádků. Celý balík tvoří 11 CIL souborů (1 326 řádků), sestavovací skript (139), Nix definice (56) a konfigurace/kontexty (154). Upstream repolicy a samostatná rozšíření jiných komponent jsou další závislosti sestavení; uvedený počet zahrnuje vlastní balík. Odečet zdrojů: **1. října 2026, 17:11 UTC**. Ověřování bezpečnostních hranic a životního cyklu pokračuje.

Tuzing — kumulativní zlepšení: model se podařilo iterativně výrazně zlepšit v porozumění pracovním dokumentům a následně ve volbě dalšího kroku. Přehled k 1. říjnu vychází z celého vývoje, s oddělenými srovnávacími sadami:

Fáze	Srovnání na stejné sadě v rámci řádku	Výsledek
Dokumentová práce, 21. září	Základní model → document-work v2	14/47 → 31/47 úspěšných úloh; sémantické skóre 0,417 → 0,808

Fáze	Srovnání na stejné sadě v rámci řádku	Výsledek
Nativní další krok, 23. září	V2 → B57	13/31 → 25/31 plně správných následujících zpráv
Další panel stejného cyklu	B57 → C87, shodné zadávací podmínky	8/18 → 13/18 ; také úspěšné dílčí editace skutečných souborů

Dokumentové úlohy používají kontrolované prostředí a modelové posouzení; nativní panely jsou malé a kurátorované. Každý řádek porovnává vlastní shodnou sadu, takže výsledky jednotlivých řádků vyhodnocujeme zvlášť. Směr další práce známe: kvalitnější příklady celého úkolu od zadání přes provedení a ověření po dodání, průběžně s kontrolou dříve získaných schopností.

Nejzajímavější je pro mě **výrazný posun malého modelu za málo peněz**. Pokud vím, k 2. říjnu mě celé dosavadní ladění stálo **méně než 200 USD**. Takové zlepšení mi dává dobrou naději, že s další investicí bude model použitelný pro zamýšlenou práci.

Dílčí náklady výpočetních pronájmů: u document-work v2 klesl kredit přibližně o **6,33 USD** během opravné diagnostické a tréninkové fáze. Nativní cyklus uzavřený 23. září spotřeboval **11,62 USD**, tedy rozdíl kreditu z 38,93 na 27,31 USD. Částka zahrnuje celé kolo; jednotlivým fitům náklady nerozdělujeme. API pro tvorbu dat a posuzování se účtovalo zvlášť.

Přehled vychází z experimentů document-work z 21. září, native-sft z 23. září a jejich souhrnu k 1. říjnu. Pozdější testy celých úloh porovnávaly dva již laděné kandidáty na jiné sadě: ve Wave A dokončili při konzervativním hodnocení 1/12 a 2/12 úloh. Výsledek závisí také na posouzení hraničních případů. Tato sada sleduje zvládnutí celého úkolu, zatímco tabulka výše ukazuje dokumentovou práci a jednotlivé další kroky.

198,49 miliardy: přesné období a sčítání

Statistika zahrnuje **čtyři Pro účty**. Vychází z výpisu denní spotřeby ze serveru z **28. září 2026** **přibližně 12:08 UTC**.

Období podle denních záznamů serveru	Tokeny	Zobrazení v miliardách
1.–31. července 2026	82 866 399 517	82,87
1.–31. srpna 2026	66 586 662 179	66,59
1.–28. září, poslední den neúplný	49 032 910 467	49,03
Celé období	198 485 972 163	198,49

Součty za kratší období vycházejí na **49,03 miliardy za září do snapshotu** a **115,62 miliardy za srpen a září do snapshotu**. Za poslední den jsou započítána data jen do okamžiku odečtu. Období se řídí daty uvedenými u jednotlivých záznamů serveru; přesná časová zóna zůstává neurčená. U jednoho účtu byl poslední denní záznam z 27. září. Do součtu vstupují záznamy čtyř Pro účtů označené datem. Historický nedatovaný rozdíl 2 141 823 tokenů zůstává vedený odděleně; stejně tak účty Plus a další poskytovatelé.

Sčítají se vstupní i výstupní tokeny včetně opakovaně zpracovaného kontextu. Vstup načtený z cache je už v celkovém počtu zahrnutý. Spotřeba MeshFleetu je součástí spotřeby těchto účtů. Koncem srpna jsem měl čtyři předplatná Pro přibližně po 200 USD měsíčně. Částka 4 423,48 USD za DeepSeek je celková útrata do konce srpna. Přibližných 100 tisíc Kč a 1,3 milionu řádků Rustu jsou údaje z mého historického experimentu TideFS.

U svých předplatných Pro za 200 USD měsíčně mám teď **kreditový rozpočet snížený na polovinu**. Tabulka zachycuje spotřebu od 1. července do 28. září.

Secrety: vlastní odmítání modelů OpenAI

Modely OpenAI samy od sebe důsledně odmítají načítat obsah secretů a při mé práci se na to mohu spolehnout.

Při procházení rolloutů v rozsahu přes 300 miliard tokenů pocházely nalezené hodnoty secretů z údajů, které jsem modelu sám výslovně předal. Model mě při jejich vložení upozorňoval na výměnu.

Praktický postup práce s přihlašovacími údaji je v kapitole 15 příručky.

Modely v kontextu mé zkušenosti

GPT Pro v chatu používám jako agentický model. „2T-class“ je moje orientační označení velikostní třídy; odhadům počtu parametrů se věnuji níže. Podle mé zkušenosti je nejschopnější dostupná AI nyní i v Codexu jako GPT 6 Astra. Při pokusech s agenty Anthropic jsem narážel i na vymyšlené výsledky testů.

Oficiální [katalog modelů OpenAI](#) a [modely v Codexu](#) k 1. říjnu 2026 uvádějí Astru i Lunu. [Dokumentace DeepSeeku](#) označuje Flash jako DeepSeek-V4.1-Flash, s API názvem deepseek-flash. Při mé práci mi Flash dává smysl zejména pro úkoly, na které bych použil OpenAI Luna.

Hypotézy o modelech a cenách

„Tick-tock“, vyšší inferenční efektivita, **lehce přes 1T / přibližně 600B / 300B parametrů** a možný cenový tlak jsou **moje spekulace**. T označuje český bilion parametrů. Vycházím ze své zkušenosti uživatele a úvahy o tom, jak firmy soutěží ve zvyšování efektivit inference. Volbu modelů při práci opírám o jejich schopnost skutečně dokončit svěřený úkol.

Moje cenová úvaha: OpenAI zatím se zlevňováním nespěchá, protože je Anthropic do něj příliš netlačí. Čekám změnu s dalším vývojem Anthropicu. S jeho modely přitom mám dál výhrady k **alignmentu**.

Kybernetické experimenty a přístup k modelům

Při práci na exploitech a red teamingu narážím na dodatečné ochrany OpenAI. Podmínky přístupu popisují [Models & Trusted Access](#) a [Cybersecurity checks](#). K 2. říjnu 2026 OpenAI rozlišuje produktové ochrany a schvalovaný program **Trusted Access for Cyber**. Vedle ověření identity je

potřeba schválení; specializovaný přístup Daybreak Red vyžaduje samostatné přidělení. Rozsah přístupu závisí také na osobě či službě, organizaci, modelu a produktu.

DeepSeek V4.1 Flash hodnotím jako dobrý model i pro tuto práci a při svých experimentech u něj na takové zábrany nenarážím.

Fieldbook v praxi na této prezentaci

Při přípravě přednášky model postupně poznává můj způsob vyjadřování. Má uložené původní vstupy a korekce, ze kterých zjišťuje, co chci říci a jak přirozeně mluvím. Díky Fieldbooku tyto souvislosti najde i čerstvý agent bez historie relace, ve které se na úkolu pracovalo, a dokáže na práci navázat.

Veřejné podklady k dalšímu čtení

- Fieldbook: <https://github.com/snajpa/fieldbook>, zejména README, AGENTS a pracovní slovník v docs/README.md.
- Snajpagent: <https://github.com/snajpa/snajpagent> a <https://agent.snajpa.net/manual.html>. Příručka dodaná s instalací popisuje její konkrétní ovládání.
- Codex CLI: <https://developers.openai.com/codex/cli/features/> a <https://developers.openai.com/codex/agent-approvals-security>.
- Práce s instrukcemi pro Codex: <https://developers.openai.com/codex/guides/agents-md>.
- OpenAI Skills: <https://developers.openai.com/codex/skills> a <https://developers.openai.com/blog/eval-skills>.
- Anthropic Agent Skills: <https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills>.
- OpenSpec: <https://github.com/Fission-AI/OpenSpec>.
- [Moje náklady a projekty](#) · [Tokeny a cena dokončené práce](#).