

מסמך מוצר מלא - LoopGuard AI

סקירה טכנית, ארכיטקטורה ופרוטוקולי הערכה

Benny Dunavich | RATIUM.AI

This is a copy of a conversation between ChatGPT & Benny Dunavich.

Report conversation

פרט לגבי המוצר loopguard-ai

Thought for 9s

בוודאי. על בסיס מה שכבר מוגדר אצלי לגבי LoopGuard AI, זהו פירוט ענייני של המוצר — בלי לייפות ובלי להמציא.

מהו LoopGuard AI

הוא מוצר שמיועד לשמש כשכבת ממשל, הערכה וקבלת החלטות עבור LoopGuard AI מערכות בינה מלאכותית, בעיקר מודלי שפה מתקדמים (LLMs) וסוכנים אוטונומיים.

הוא לא מתיימר להיות המודל עצמו, אלא שכבה שמעל או לצד המודל, שמטרתה לענות על השאלה:

האם מותר לשחרר / להפעיל / להמשיך / להגביל / לעצור את המערכת — ועל בסיס מה?

במילים פשוטות:

אם המודל הוא "המנוע", אז LoopGuard AI הוא מערכת הבקרה, המדידה והשערים.

הבעיה שהמוצר בא לפתור

הנחת המוצא של LoopGuard AI היא שבעולם LLM לא מספיק לשאול:

האם המודל "חכם"

האם הוא ענה יפה

האם הוא עבר benchmark

אלא צריך לשאול גם:

האם ההתנהגות שלו עקבית

האם יש סימני סיכון או הידרדרות

האם יש הצדקה מוסברת להחלטת שחרור

האם אפשר לשחזר את ההחלטה

האם הארגון מחזיק evidence מסודר מול רגולציה, ביקורת, לקוח או הנהלה

כלומר, המוצר מכוון לבעיה שבין evaluation לבין governance:

לא רק למדוד, אלא גם לתרגם מדידות להחלטת פעולה תפעולית.

מה התוצר המרכזי של המערכת

לפי הפרופיל שכבר הוגדר, LoopGuard AI אמור להפיק החלטות שער מוסברות, בסגנון:

אפשר לשחרר — SHIP

אפשר להפעיל תחת הגבלות — RESTRICT

לעצור ולבדוק — HOLD

לחזור לגרסה קודמת — ROLLBACK

זו נקודה חשובה:

המוצר לא רק "מראה דוח", אלא שואף להיות שכבה שמאפשרת decision-ready output.

איך המוצר בנוי ברמה עקרונית

מה שכבר מוגדר לגבי המוצר מצביע על ארכיטקטורה מסוג:

1. שכבת קלטים

המערכת אוספת או מקבלת:

פלטי מודל

תוצאות evals

מבחני איכות

תרחישי jailbreak / policy

בדיקות drift / consistency

מידע על גרסה, פרומפטים, קונפיגורציה, ספק מודל, סביבת ריצה

כלומר, היא לא חייבת להחליף את כלי ה-evaluation הקיימים; היא יכולה לשבת מעליהם או לצדם.

2. שכבת חישוב מדדים

כאן LoopGuard AI מחשב מדדי סיכון/איכות/עקביות/ממשל.

בפרופיל שלך הודגשו במיוחד:

NFCI

Core / Shell

אלו אינם מדדי benchmark סטנדרטיים בתעשייה, אלא שכבת מדידה מובחנת של המוצר.

3. שכבת מדיניות ושערים

המערכת ממפה את המדדים לכללי החלטה:

מה עובר

מה נכשל

מה דורש אדם בלולאה

מה אסור לשחרר

4. שכבת הסבר וביקורת

המערכת שומרת:

audit trail

הסבר להחלטה

evidence exportable

מסלול שחזור של ההכרעה

כאן טמון חלק מהבידול: לא רק "score", אלא traceable decision.

מה המשמעות של Core / Shell

על בסיס מה שכבר הוגדר אצלך, ההבחנה בין Core ל-Shell היא חלק מזהות המוצר.

באופן תפעולי, אפשר להבין זאת כך:

שכבת הסימפטומים, הביטוי הגלוי, האותות החיצוניים, התנהגות נראית — Shell

שכבת העומק: דפוסי כשלים, חוסר עקביות עקרוני, מנגנוני סיכון עמוקים יותר — Core

במונחי מוצר, הרעיון הוא שלא מספיק לזהות "מה יצא", אלא צריך לנסות להבחין בין:

כשל נקודתי

לבין בעיית עומק שחוזרת על עצמה

זה בדיוק סוג הדברים שיכולים להפוך הערכה חד-פעמית ל-governance discipline.

למה NFCI חשוב במוצר

כפי שהוא מוגדר במארג שלך, הוא אחד ממדדי הליבה שמטרתם להעריך איכות של NFCI, שפה מוסדית/נרטיבית/קוגניטיבית ולהפוך אותה למדד תפעולי.

במסגרת LoopGuard AI, התפקיד של מדד כזה הוא לא רק "להיות מעניין פילוסופית", אלא לשמש כעוד ציר הערכה בהחלטה אם מערכת:

יציבה

נשלטת

עקבית

מסבירה את עצמה

מסוכנת מבחינת דפוסי סטייה

זהו רכיב מבדל, אבל גם רכיב שידרוש בעתיד הוכחה אמפירית חזקה, אחרת הוא ייראה כמו מילת קסם מחופשת למדד. והעולם מלא במילות קסם כאלה, למרבה העליצות הבירוקרטית.

מי הלקוחות הפוטנציאליים

לפי הדרך שבה המוצר כבר הוגדר, הוא מתאים בעיקר ל:

ספקי AI / LLM

חברות שבונות או מטמיעות מודלים ורוצות שכבת החלטה לפני שחרור.

ארגונים אנטרפרייז

גופים שמתמשים במודלי שפה בתהליכים רגישים וצריכים:

בקה

תיעוד

traceability

שפה של compliance

צוותי AI Governance / Risk / Compliance

כאלה שרוצים לעבור ממצב של "יש לנו הרבה evals" למצב של "יש לנו מנגנון החלטה".

צוותי מוצר והנדסה

כאלה שצריכים לקבוע:

האם גרסה חדשה טובה מספיק לשחרור או לא.

מה הממשק האפשרי עם תשתיות קיימות

לפי ההגדרה שלך, לא חייב להיות מערכת סגורה, LoopGuard AI,

הוא יכול לשבת כִּי:

SDK

Dashboard

ולעתים גם on-prem deployment

כלומר, הוא עשוי להשתלב עם stack קיים של release-ו eval, observability, policy checks workflow.

הערך כאן הוא לא "עוד dashboard", אלא ניסיון להיות שכבת orchestration של הכרעה.

מה הבידול של המוצר לעומת כלי evaluation רגילים

הבידול המרכזי, לפי מה שמוגדר, הוא בצירוף של כמה דברים

יחד:

1. מעבר מ- evaluation ל- decision

רבים מודדים. פחות יודעים להכריע.

2. שערי החלטה ברורים

לא רק ציון, אלא תוצאה תפעולית.

3. auditability

יכולת להסביר למה התקבלה החלטה.

4. שכבת metrics מקורית

ושפה אנליטית מובחנת NFCI, Core/Shell

5. זיקה למסגרות ממשל וציות

כבר הוגדר אצלך כיוון ליישור מול:

NIST AI RMF

ISO/IEC 42001

ונרטיב תאימות אירופי

חשוב להדגיש:

זה לא אומר שהמוצר כבר הוכח כתואם רגולטורית במובן פורמלי.
זה כן אומר שהוא ממוצב כמוצר שיכול לייצר compliance narrative + evidence trail.
מה הצעת הערך העסקית

מבחינה עסקית, ההיגיון של המוצר הוא:

להפחית סיכון של שחרור גרסה בעייתית
לקצר זמן עד החלטה
להפחית תלות בשיקול דעת ידני מפוזר
להגדיל עקביות בין צוותים
לשפר יכולת ביקורת, תיעוד ושחזור
להפוך chaos של evals ל-workflow מסודר

כלומר, הלקוח לא קונה רק "בדיקה", אלא:

שכבת שליטה ארגונית על החלטות AI.
מה מצב הבשלות כפי שעולה ממה שכבר הוגדר

כאן צריך להיות ישר:

על בסיס מה שקיים בזיכרון, LoopGuard AI מתואר היטב ברמה מושגית, ארכיטקטונית ומוצרית, אבל אין אצלי כרגע בסיס לומר שהמוצר כבר הוכח אמפירית כמערכת מסחרית בשלה.

בפרט, לא הוצגו לי עד כה כראיות סגורות:

מלאים benchmark results
baseline comparisons
תוצאות POC מתועדות

precision/recall של gate decisions

false positive / false negative rates

מלא reproducibility pack

לקוחות פעילים או deployment מוכח

לכן, נכון יותר לומר:

יש כיום:

מסגרת מוצרית חזקה

רעיון מובחן

שפה תיאורית ואסטרטגית עשירה

governance/evaluation מעניין מול positioning

עדיין חסר, כדי להפוך את זה למוצר מוכח:

evidentiary backbone

ניסויי שדה

baseline comparison

הנדסיים מוחשיים artifacts

הגדרת integration path מדויקת מול stack קיים

מה עשוי להרשים קהילת AI כבר עכשיו

גם בלי להעמיד פנים שיש הוכחה מלאה, יש כמה דברים

שבכוחם לעורר עניין:

1. המסגור

המעבר מ"evaluation report" ל-"release decision system".

2. שכבת governance ייעודית

זהו צורך אמיתי בשוק.

3. החיבור בין מדידה, שערים, `audit trail` ו-`compliance`

זה ציר מעניין ומעשי.

4. הניסיון להציע `metrics` משלך

זה מבדל, בתנאי שלא נשארים ברמת סיסמה.

מהו הסיכון המרכזי של המוצר

הסיכון המרכזי הוא שהמוצר יתפס כ:

שפה חזקה בלי הוכחת ביצועים מספקת.

כלומר, אם לא יהיה ברור:

איך מחשבים את המדדים

מה איכותם

מול איזה `baseline`

באילו תרחישים הם מועילים

איך מוכיחים שהם משפרים החלטה

אז גם רעיון טוב יישאר בגדר מסגרת אינטלקטואלית ולא יהפוך לכלי שמקבל תקציב.

איך נכון להציג את המוצר כרגע

הניסוח הכי מדויק, לדעתי, הוא משהו בסגנון:

וסוכנים, שמטרתה להמיר LLMs עבור `Governance & Evaluation` הוא שכבת `LoopGuard AI` מדדי איכות/סיכון/עקביות להחלטות שער מוסברות, עם `audit trail` ויכולת השתלבות בתהליכי `release` ובציות.

זה ניסוח טוב כי הוא:

לא מבטיח ניסים

לא מציג "AGI"

כן מציג ערך ברור

כן מחדד את ההבדל מול eval בלבד

סיכום חד

הוא, לפי ההגדרה הקיימת, מוצר שמנסה לפתור את הבעיה שבין LoopGuard AI

"האם המודל נבדק?" לבין

"האם מותר לנו לשחרר אותו?"

הליבה שלו היא:

metrics

governance

decision gates

explainability

auditability

compliance-ready framing

החוזקה שלו כרגע היא בעיקר במסגור המוצר והארכיטקטורה העקרונית.

החולשה המרכזית כרגע היא מחסור בראיות תפעוליות מוכחות.

אם תרצה, אעבור עכשיו לשלב הבא ואפרט אותו באחת משלוש רמות:

טכנית, עסקית, או כמסמך הצגה לשותף/משקיע.

Thought for 59s

להלן מסמך ארכיטקטורה הנדסי מלא עבור LoopGuard AI — בנוסח עבודה טכני, ממושמע, ובלוי להעמיד פנים שהיכן שאין עדיין ראיות יש "מוצר גמור".

כל מה שמסומן כאן כ־מומלץ / מוצע הוא תכנון הנדסי. כל מה שלא הוכח בארטיפקטים, לוגים או ניסויי שדה — אינו נחשב כאן לממומש.

LoopGuard AI

מסמך ארכיטקטורה הנדסי מלא

גרסת עבודה: 0.1

סטטוס: Reference Architecture / Design Spec

ייעוד: Governance & Evaluation Layer עבור LLMs ו־Agent Systems

1. מטרת המערכת

היא שכבת תוכנה שמטרתה להמיר אותות הערכה, סיכון, איכות, מדיניות LoopGuard AI ועקביות של מערכת בינה מלאכותית להחלטת שער תפעולית, מוסברת, ניתנת לשחזור ומתועדת.

המערכת נועדה לענות על השאלה:

האם מותר לשחרר / להגביל / לעצור / להחזיר גרסה של מערכת AI — ועל בסיס אילו ממצאים?

המערכת אינה מחליפה את המודל, ואינה מחליפה בהכרח את כלי ה־evaluation הקיימים. תפקידה הוא להיות שכבת orchestration, scoring, gating, audit and evidence.

2. מטרות־על טכניות

המוצר נדרש לספק חמש יכולות יסוד:

קליטת אותות ממקורות מרובים: evals, logs, policy checks, regression suites, red-team metadata.

נרמול כל הקלטים למודל נתונים אחיד.

חישוב מדדים פרימיטיביים, נגזרים ומבניים.

הכרעת שער באמצעות מנוע חוקים/מדיניות.

הפקת evidence package מלא: rationale, thresholds, failed scenarios, overrides, audit trail.

3. גבולות המערכת

בתוך תחום האחריות

Pre-release gating

Post-release review and rollback recommendation

Governance of LLM and agent runs

Risk/quality scoring

Decision traceability

Exportable compliance evidence

מחוץ לתחום, לפחות ב-MVP

של המודל עצמו Training

Fine-tuning orchestration

Full observability replacement

Full SOC/GRC replacement

Real-time token-by-token moderation

Autonomous runtime blocking of every agent action

הערה קריטית:

ב-MVP, המערכת צריכה להיבנות קודם כול כ-runtime batch / async decision system. מלא הוא שלב מאוחר יותר. אחרת מקבלים מפלצת ארכיטקטונית לפני שיש enforcement רגליים.

4. עקרונות תכנון מחייבים

כל החלטה חייבת להיות מוסברת — Explainability first

אותה ⇒ metric אותה גרסת, policy אותה, inputs אותם — Determinism where possible החלטה.

Version everything — metric definitions, policy packs, schemas, calibration sets.

Pluggable metrics — אפשר להוסיף או להחליף מנועי מדידה

Tenant isolation — הפרדה מלאה בין לקוחות/סביבות

Audit by design — audit trail הוא רכיב ליבה

Human override with trace — override מתועד, אך חייב להיות מתועד

Fail closed for critical policy paths — לא משחררים כאילו בסדר

5. מקרי שימוש עיקריים

5.1 שער שחרור לפני deploy

מערכת eval קיימת מספקת תוצאות. LoopGuard AI מחליט:

SHIP

RESTRICT

HOLD

5.2 בדיקת regression בין גרסאות

השוואת build חדש לעומת baseline מאושר.

5.3 בקרת agent workflow

בדיקת מסלול פעולה של agent:

בחירת tools

סטיות תכנון־ביצוע

autonomy risk

escalation behavior

5.4 post-release review

ניתוח אירועי פרודקשן והמלצת ROLLBACK.

5.5 evidence export

הפקת חבילת ראיות עבור הנהלה, לקוח, ביקורת או רגולציה.

6. טופולוגיית המערכת

יש להבחין בין שלוש תצורות פריסה:

6.1 SaaS

שליטה נוחה, onboarding מהיר, מתאים ללקוחות לא-רגישים.

6.2 Hybrid

של הלקוח VPC מקומית או ב־ artifacts, בענן Control plane

6.3 On-prem

חובה עבור לקוחות regulated או רגישים.

המלצה הנדסית

להגדיר את המוצר מלכתחילה כ־ deployment-agnostic, עם אותם contracts לוגיים בכל מצב.

הבדלים בפריסה לא צריכים לשבור את מודל הנתונים או את מנוע ההכרעה.

7. מבט־על ארכיטקטוני

הארכיטקטורה הלוגית המוצעת:

[Sources / Connectors]

|

v

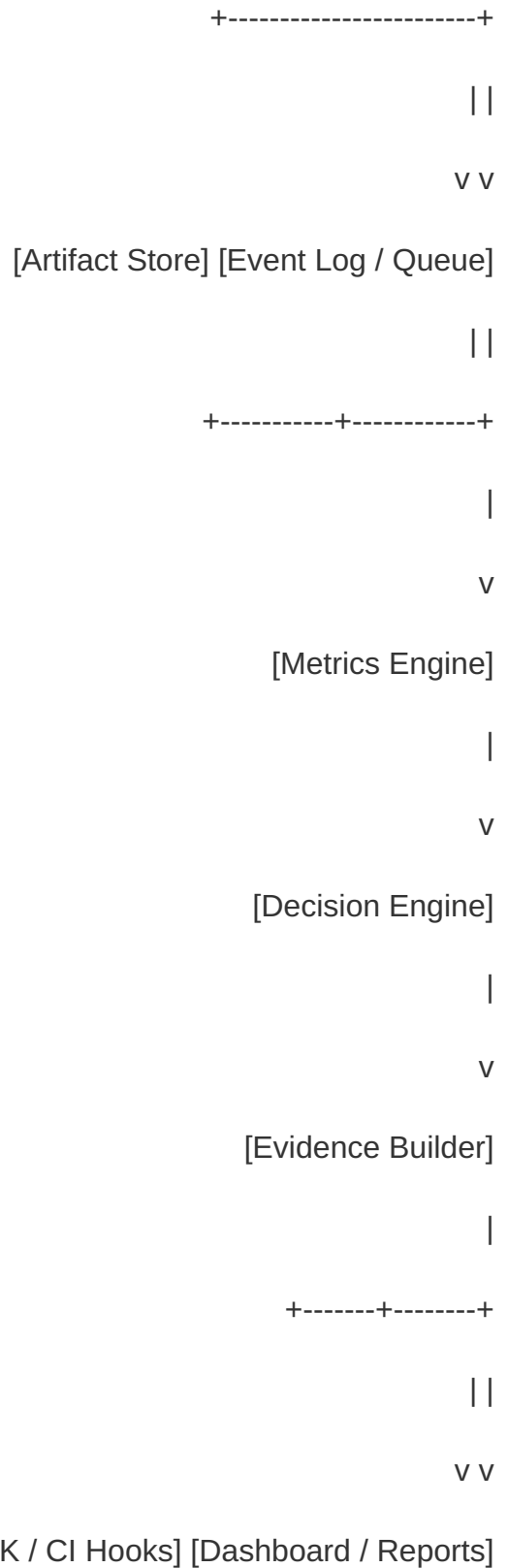
[Ingestion Layer]

|

v

[Normalization & Canonical Schema]

|



8. רכיבי המערכת

8.1 Connectors / Sources

שכבת חיבורים למקורות חיצוניים.

סוגי מקורות:

Eval frameworks

Red-team outputs

Prompt/response logs

Agent traces

Model metadata

Policy scanners

Manual reviewer inputs

CI/CD pipelines

אחריות:

קבלת נתונים בפורמטים שונים

אימות בסיסי

ראשוני tagging

timestamping

source attribution

לאחריות שכבה זו לא שייך scoring מהותי.

8.2 Ingestion Layer

אוסף קלטים ומעביר אותם למבנה עבודה אחיד.

תכונות נדרשות:

idempotency

retry handling

schema validation

checksum/hash for artifacts

batch + streaming ingestion

פלט:

canonical records

artifact references

ingestion events

8.3 Normalization & Canonical Schema

שכבה זו קריטית. בלי schema קנוני, אין מוצר — יש אוסף adapters עצבני.

המערכת צריכה להחזיק מודל נתונים אחיד עבור:

Run

Scenario

Signal

Metric

Rule evaluation

Decision

Override

Evidence bundle

8.4 Artifact Store

אחסון בלתי-מובנה של:

raw eval outputs

logs

prompt/response pairs

screenshots

exported reports

red-team artifacts

trace snapshots

המלצה:

לראיות חתומות עם immutable mode Object storage

hash-based addressing

retention policy לפי policy pack / tenant / regulation tier

8.5 Event Log / Queue

שכתב אירועים פנימית לצורך:

metrics-ל ingestion בין decoupling

reprocessing

observability

replay

audit reconstruction

סוגי אירועים:

RunCreated

SignalIngested

MetricComputed

RuleTriggered

DecisionIssued

OverrideApplied

EvidenceGenerated

RunClosed

8.6 Metrics Engine

מנוע חישוב המדדים.

שלוש משפחות מדדים

Primitive metrics — pass rates, failure rates, latency, consistency variance.

Derived metrics — risk score, drift severity, regression delta, escalation rate.

Structural metrics — Core, Shell, NFCI.

דרישת תכן מחייבת

כל metric חייב להחזיק:

name

version

inputs used

formula or inference method

calibration reference

confidence / completeness flag

explainability note

בלי versioning של metric, מתחיל קרקס: אותה מילה, שני חישובים, שלוש מסקנות.

8.7 Decision Engine

לב המערכת.

קלט:

metric results של vector

policy pack

environment context

deployment risk tier

baseline comparison

human review flags

פלט:

SHIP

RESTRICT

HOLD

ROLLBACK

יחד עם:

rationale

triggered rules

blocking evidence

recommended actions

escalation requirement

8.8 Policy Registry

מאגר policy packs גרסאי.

מגדיר Policy pack:

thresholds

precedence rules

mandatory blockers

tenant overrides

environment-specific tolerances

domain rules

evidence requirements

retention rules

דוגמאות:

low-risk internal chatbot

enterprise customer support

regulated domain assistant

agent with tool access

8.9 Evidence Builder

מייצר חבילות ראיות הניתנות לייצוא.

חבילת evidence כוללת:

run metadata

model version

inputs summary

metrics summary

rule trace

decision rationale

artifact refs

reviewer overrides

timestamps

signature / integrity hash

8.10 API / SDK / Dashboard

ממשקי גישה למערכת.

API

לשילוב טכני עם מערכות.

SDK

להטמעה פשוטה בתוך pipelines.

Dashboard

להצגת:

current runs

failed gates

drift trends

decision trace

policy health

evidence exports

8.11 Identity / Access / Admin

נדרש מנגנון:

RBAC

audit of admin changes

per-tenant policy controls

approval workflows

secret management

מדדי מערכת פנימיים:

ingestion lag

metric computation failures

queue depth

rule execution latency

decision issuance latency

evidence generation failures

per-tenant error rate

9. מודל הנתונים הקנוני

להלן הישויות המרכזיות.

9.1 Run

מייצג יחידת הערכה אחת.

שדות נדרשים:

run_id

tenant_id

environment (dev/staging/prod)

model_id

model_version

system_prompt_version

dataset_version

policy_pack_version

risk_tier

created_at

status

סטטוסים |

CREATED

INGESTING

SCORING

DECIDING

DECIDED

OVERRIDDEN

CLOSED

FAILED

9.2 Scenario

תרחיש בדיקה ספציפי.

שדות |

scenario_id

run_id

scenario_type

prompt_ref

expected_behavior

policy_tags

severity_weight

9.3 Signal

אות גולמי או מנורמל.

שדות:

signal_id

run_id

scenario_id

source_type

source_ref

payload_ref

raw_score

normalized_score

timestamp

9.4 MetricResult

תוצאת חישוב metric.

שדות:

metric_name

metric_version

scope (scenario/run/version/global)

value

confidence

inputs_hash

computation_ref

explainability_note

9.5 RuleEvaluation

תוצאת הרצת כלל.

שדות:

rule_id

rule_version

triggered (bool)

severity

inputs_used

reason_text

9.6 DecisionPackage

פלט ההכרעה.

שדות:

decision_id

run_id

final_decision

confidence

blocking_rules

summary

required_actions

issued_at

9.7 Override

תיעוד החלטת אדם.

שדות:

override_id

decision_id

actor_id

old_decision

new_decision

justification

approved_by

timestamp

9.8 EvidenceBundle

חבילת ראיות חתומה.

שדות:

bundle_id

run_id

artifact_refs

summary_ref

integrity_hash

generated_at

10. זרימות עבודה

10.1 Pre-release Gate Flow

1. CI/CD or Eval System opens Run

2. Connectors push signals/artifacts
 3. Ingestion validates and normalizes
 4. Metrics Engine computes primitive metrics
 5. Derived metrics and structural metrics computed
 6. Decision Engine loads relevant policy pack
 7. Rules executed in precedence order
 8. Decision package created
 9. Evidence bundle generated
 10. API returns SHIP / RESTRICT / HOLD
 11. CI/CD acts accordingly
- 10.2 Post-release Incident Flow
1. Production incident or anomaly detected
 2. Relevant traces/logs ingested
 3. Delta vs approved baseline calculated
 4. Critical rules re-evaluated
5. Decision Engine may recommend HOLD or ROLLBACK
6. Human review may override with trace
 7. Evidence package stored
- 10.3 Agent Evaluation Flow
1. Agent trace ingested (plan, tool calls, outputs)
 2. Sequence features extracted
 3. Metrics computed on trajectory, not only final output
 4. Unsafe autonomy / divergence / unauthorized tools checked

5. Final decision emitted with trajectory rationale

11. מנוע המדדים

11.1 Primitive Metrics

מדדים בסיסיים ישירים.

דוגמאות:

pass_rate

fail_rate

refusal_correctness

hallucination_indicator

latency_p95

consistency_variance

tool_call_error_rate

policy_violation_count

11.2 Derived Metrics

מחושבים משילוב primitive metrics.

דוגמאות:

release_risk_score

regression_severity

drift_delta

escalation_rate

decision_confidence

baseline_gap

11.3 Structural Metrics

כאן יושב הבידול של המוצר.

Core

מטרתו להעריך עומק מבני של בעיה:

pattern recurrence

cross-scenario failure coherence

deep inconsistency signals

severity under variant prompts or contexts

Shell

מטרתו להעריך ביטוי חיצוני/תפעולי:

frequency

spread

visibility across scenarios

outward manifestation severity

NFCI

כאן נדרשת זהירות כפולה.

תפקודו הטכני המוצע:

להעריך איכות מבנה הסבר/שפה/נימוק/עקביות מוסדית או קוגניטיבית

לשמש כציר תומך בהחלטת סיכון ולא כתחליף לכל שאר המדדים

הערת משמעת

בשלב זה, בלי spec חישובי מלא, Core / Shell / NFCI הם שמות של interfaces מדידתיים, לא עדיין נוסחאות מוכחות.

כלומר: יש מקום בארכיטקטורה; נדרש עדיין formal metric spec.

11.4 Metric Plug-in Interface

כל metric צריך להיות ממומש כפלאגין עם חוזה אחיד:

Input:

- canonical signals
- artifacts refs
- run context
- calibration config

Output:

- value
- confidence
- explanation
- evidence refs
- failures / incompleteness

11.5 Calibration and Validation

לכל metric נדרשים:

gold / silver labeled examples

baseline comparisons

drift monitoring of the metric itself

versioned threshold calibration

reproducibility checks

מדד שאי אפשר לכייל, לשחזר ולהסביר — הוא קישוט תיאורטי, לא רכיב מוצר.

12. מנוע ההחלטה

12.1 מודל ההכרעה

מנוע ההחלטה הוא rule-based בראש ובראשונה, עם אפשרות צמיחה לרכיבי scoring סטטיסטיים.

המלצה: לא להתחיל במנוע קופסה שחורה. עולם הממשל הארגוני לא אוהב שופט מסתורי.

12.2 היררכיית כללים

סדר עדיפות מוצע:

Hard blockers

policy critical failures

missing critical evidence

catastrophic regression

forbidden tool usage

Risk elevation rules

מעל סף Core

repeated multi-scenario failures

severe drift vs baseline

Conditional restriction rules

medium-risk instability

incomplete evidence

model-specific caveats

Green path

all mandatory checks passed

no hard blocker

acceptable drift

evidence complete

12.3 תוצאות השער

SHIP — מותר שחרור

tenant מסוים מותר שחרור מוגבל, למשל רק לדומיין או — RESTRICT

HOLD — עצירה עד בדיקה נוספת

ROLLBACK — חזרה לגרסה קודמת או הקפאת גרסה פעילה

12.4 Override Logic

אנושי מותר רק עם override

justification

approver role

time-bound validity

full audit trace

אין "לחצתי כי נראה לי". זה מתכון מעולה לכאוס עם חליפה.

13. Policy Packs

הוא אובייקט גרסאי שמחזיק Policy pack

threshold definitions

required metrics

blocker rules

allowed overrides

evidence minimums

retention configuration

דוגמה רעיונית:

Policy Pack: Enterprise-Regulated-v3

- Hard block on critical policy failures
- Hold if Core ≥ 0.62
- Restrict if drift_delta > 0.15 and no human signoff
- Rollback recommendation if post-release catastrophic regression \geq threshold
- Evidence completeness must be $\geq 95\%$

הערה: המספרים כאן illustrational בלבד, לא claim.

מוצע API 14.

14.1 פתיחת ריצה

POST /runs

קלט:

tenant

model version

environment

policy pack version

metadata

פלט:

run_id

14.2 הזרמת אותות

POST /runs/{run_id}/signals

signal batch

source metadata

artifact refs

artifacts העלאת 14.3

POST /runs/{run_id}/artifacts

metrics חישוב 14.4

POST /runs/{run_id}/score

קבלת החלטה 14.5

POST /runs/{run_id}/decide

שליפת החלטה 14.6

GET /runs/{run_id}/decision

evidence שליפת 14.7

GET /runs/{run_id}/evidence

14.8 override

POST /runs/{run_id}/override

14.9 policy packs

GET /policies

GET /policies/{version}

14.10 health / readiness

GET /health

GET /ready

15. SDK מוצע

ה-SDK צריך לאפשר:

- create run
- attach metadata
- push batch signals
- upload artifact refs
- trigger scoring/decision
- fetch decision
- attach override
- CI/CD integration helpers

שפות יעד מומלצות ל-SDK הראשון:

- Python
- TypeScript
- 16. שכבת האחסון
- 16.1 Metadata Store
- המלצה: RDBMS כמו PostgreSQL.

מתאים ל:

- runs
- metrics
- decisions
- policies
- overrides
- audit metadata

16.2 Artifact Store

המלצה: object storage תואם S3.

מתאים ל:

raw outputs

logs

reports

snapshots

16.3 Event Transport

המלצה: queue או streaming layer.

אפשרויות:

Kafka / Redpanda

RabbitMQ

cloud queue equivalent

הבחירה תלויה בהיקף. ב-MVP אפשר להתחיל פשוט יותר.

16.4 Cache

לשימוש נקודתי:

policy pack cache

recent decision lookup

dashboard acceleration

17. אבטחת מידע ופרטיות

17.1 עקרונות

least privilege

tenant isolation

encryption in transit and at rest

signed evidence bundles

immutable audit records

secrets management

PII minimization

17.2 מודל גישה

תפקידים:

Admin

Policy Admin

Reviewer

Operator

Read-only Auditor

CI Service Account

17.3 Data Sensitivity

המערכת צריכה לתמוך במצבים:

no raw prompts retained

hashed-only mode

full trace retention

redacted evidence bundles

17.4 On-prem readiness

local object store support

external secrets manager integration

private networking

offline policy execution

18. אמיונות, ביצועים ו-SLO

18.1 יעדי מערכת מוצעים

יצירת run: תת־שנייה עד שניות בודדות

שניות עד דקות, לפי נפח: scoring batch

scoring תוספת מינימלית מעל: decision issuance

אסינכרוני מועדף: evidence generation

18.2 זמינות

כיעד סביר 99.9%: Control plane

Decision critical path: degradation-aware

audit store: strongly durable

18.3 failure modes

connector down

partial artifact loss

metric plugin failure

queue lag

policy pack mismatch

incomplete evidence

timeout in scoring

בכשל מהותי:

mark run as incomplete

prevent false SHIP

expose error cause clearly

19. שחזור, עקביות וגרסאות

כל החלטה חייבת להיות קשורה ל:

metric versions

policy version

run metadata hash

artifact hash set

rule engine version

calibration reference

תנאי שחזור:

אם מריצים שוב אותו run עם אותם inputs, אותה policy ואותן גרסאות — ההחלטה צריכה להיות זהה, או לכל הפחות להציג deviation reason מפורש.

20. UI / Dashboard

ה־dashboard צריך לכלול לפחות:

20.1 Run View

status

model version

environment

decision

blocking rules

failed scenarios

20.2 Decision Trace View

metrics

thresholds

triggered rules

rationale

evidence links

20.3 Trend View

release outcomes over time

drift by model version

override frequency

policy violation distribution

20.4 Audit View

who changed what

when

why

which policy version

21. אינטגרציות עיקריות

21.1 CI/CD

GitHub Actions

GitLab CI

Jenkins

Argo / internal release orchestration

21.2 Eval frameworks

generic JSON/CSV ingestion

custom adapters

benchmark results import

21.3 LLM orchestration stacks

gateways

prompt routers

agent platforms

21.4 Model providers

model metadata integration

version tagging

per-provider risk configuration

22. ארכיטקטורת MVP לעומת Target Architecture

22.1 MVP

מומלץ לכלול:

async/batch only

run ingestion

canonical schema

primitive + derived metrics

placeholder interface for Core/Shell/NFCI

rule-based decision engine

evidence export

בסיסי dashboard

CI/CD hook

22.2 שלב 2

baseline comparison engine

richer policy packs

agent trace support

manual review workflows

better calibration tooling

22.3 שלב 3

post-release rollback assist

real-time signals

advanced sequence analysis

tenant self-service policy management

23. טכנולוגיות ייחוס מומלצות

אלה המלצות, לא תיאור מצב קיים מוכח.

Backend API: Python/FastAPI או Go

Metrics/Decision services: Python

Metadata DB: PostgreSQL

Artifact store: S3-compatible

Queue: Kafka/Redpanda או RabbitMQ

Dashboard: React/Next.js

Auth: OIDC/SAML

Infra: Kubernetes

היגיון הבחירה:

AI של ecosystem ואינטגרציה עם metrics, evals ל־ Python.

טוב יותר אם הדגש יעבור לקשיחות תפעולית Go.

ב־Python, MVP הוא הנתיב הפשוט יותר.

24. בדיקות הנדסיות נדרשות

24.1 Unit tests

schema validation

metric plugins

rule evaluation

evidence assembly

24.2 Integration tests

connector → ingestion → scoring → decision

artifact reference integrity

policy pack loading

24.3 Reproducibility tests

same input → same decision

version shift → expected diff

24.4 Adversarial tests

malformed artifacts

missing metadata

duplicate ingestion

delayed signals

inconsistent timestamps

24.5 Security tests

tenant escape

privilege escalation

tampered evidence

replay attacks

25. הנדסיים למוצר עצמו KPI

LoopGuard AI גם את עצמו צריך למדוד.

decision latency

percentage of explainable decisions

evidence completeness rate

override rate

false hold rate

missed critical failure rate

metric computation success rate

reproducibility pass rate

26. סיכונים מרכזיים

26.1 Metric formalization risk

עלולים להישאר עמומים מדי Core / Shell / NFCI.

26.2 Baseline risk

בלי baseline טוב, קשה להוכיח תועלת.

26.3 Evidence burden

אם הלקוח צריך להזרים יותר מדי חומר ידנית — adoption ייפגע.

26.4 Explainability gap

אם המנוע ייעשה מסובך מדי, לקוחות regulated יירתעו.

26.5 Runtime overreach

ניסיון לקפוץ מוקדם מדי ל־runtime enforcement יפיל את המוצר להנדסת־יתר.

27. שאלות פתוחות שמחייבות הכרעה

האם MVP כולל רק pre-release, או גם post-release?

האם Core / Shell / NFCI יחושבו rule-based, classifier-based, או hybrid?

מהו baseline canonical הראשון?

מהו policy pack הראשון שייחשב reference customer-ready?

האם evidence bundles יהיו JSON-first או PDF-first?

מה רמת ה־agent support בגרסה הראשונה?

מהו מודל ה־tenant isolation באונפרם?

28. קריטריוני קבלה לגרסה ראשונה

גרסה ראשונה נחשבת בשלה רק אם מתקיימים יחד:

אפשר לפתוח run ולהזרים signals/artifacts.

המערכת מחשבת metrics גרסאיים.

מנוע ההכרעה מחזיר SHIP / RESTRICT / HOLD בצורה דטרמיניסטית.

יש decision trace מלא.

יש evidence bundle חתום.

ניתן לבצע override אנושי מתועד.

ניתן לשלב את המערכת ב־CI/CD פשוט.

יש לפחות baseline demo אחד עם repeatable results.

אין SHIP false במקרי blocker ידועים.

ה-API וה-schema מתועדים.

29. פסקידין הנדסי

כמסמך ארכיטקטורה, LoopGuard AI יכול להיות מוגדר באופן רציני כמערכת בת 8 שכבות:

Connectors

Ingestion

Canonical Schema

Artifact Store / Event Log

Metrics Engine

Decision Engine

Evidence Builder

API / SDK / Dashboard

החוזקה הארכיטקטונית שלו היא ברעיון שלו:

signals → normalized evidence → metrics → decision → audit

נקודת החולשה הנוכחית, מבחינה הנדסית, היא לא בארכיטקטורה עצמה אלא בפער בין השמות של המדדים הייחודיים לבין פורמליזציה מוכחת שלהם.

במילים ישירות:

השלד ההנדסי ניתן לבנייה.

המאבק האמיתי יהיה על הגדרת baselines, calibration, metric spec, ו-evidence pack.

30. נוסח קצר מנהלי לסוף המסמך

המאגדת LLMs ו-Agents עבור Governance & Evaluation מתוכנן כשכבת LoopGuard AI אותות הערכה ממערכות חיצוניות, ממירה אותם למודל נתונים אחיד, מחשבת מדדי סיכון/איכות/מבנה, מפעילה מנוע החלטה מבוסס policy, ומפיקה החלטת שער מוסברת עם evidence export - audit trail.

ה-MVP ההנדסי הנכון הוא מערכת pre-release determinist batch עם schema קנוני, evidence bundles, rule-based decision engine, metrics versioning.

האתגר הקריטי אינו בהקמת ה-plumbing אלא בהוכחת תוקף ושחזור של המדדים הייחודיים.

השלב הבא ההנדסי המתבקש הוא להמיר את המסמך הזה ל-System Design Spec עם שלושה נספחים מחייבים: OpenAPI ראשוני, סכמת DB ראשונית, ו-Metric Spec v0.1 ל-Core/Shell/NFCI.

רשום פרטי מפתח loopguard-ai

Thought for 14s

להלן פרטי מפתח של LoopGuard AI בנוסח מרוכז:

זהות המוצר

עבור מודלי שפה Governance & Evaluation הוא מוצר שמיועד להיות שכבת LoopGuard AI מתקדמים (LLMs) ו-Agent Systems.

הוא לא מחליף את המודל עצמו, אלא יושב מעל או לצד המודל כדי להכריע אם מותר לשחרר / להגביל / לעצור / להחזיר גרסה.

הבעיה שהמוצר פותר

הבעיה המרכזית היא הפער בין:

יש תוצאות בדיקה — Eval

לבין

האם מותר לשחרר לפרודקשן — Decision

כלומר, המוצר נועד להפוך אוסף בדיקות, לוגים, flags וציוני סיכון ל־החלטת שער תפעולית מוסברת.

פלט הליבה

המוצר אמור להחזיר אחת מארבע החלטות:

SHIP

RESTRICT

HOLD

ROLLBACK

בצירוף:

נימוק

כללים שהופעלו

evidence

audit trail

אנושי אם היה override

רכיבי הליבה הארכיטקטוניים

המבנה ההנדסי שהוגדר לו כולל:

Signal Ingestion

Normalization / Canonical Schema

Metrics Engine

Decision Engine

Evidence Builder

API / SDK / Dashboard

סוגי הקלטים

אמור לעבוד עם LoopGuard AI:

תוצאות eval

prompt/response pairs

red-team / jailbreak tests

policy checks

regression / drift signals

metadata של גרסה, prompt, dataset, environment

וכלי עזר של agents של traces

המדדים המובחנים של המוצר

בפרופיל המוצר הודגשו במיוחד:

Core

Shell

NFCI

המשמעות התפעולית:

הביטוי החיצוני של הבעיה — Shell

עומק/מבניות של הכשל — Core

מדד ייחודי נוסף שאמור לתרום להערכת איכות/עקביות/מבנה נימוק או שפה — NFCI

הערה קריטית:

נכון לעכשיו, יש למסגרת הזו זהות מושגית חזקה, אבל לא הוצגו עדיין מספיק ראיות לכך שהמדדים הללו כבר נוסחו, כוילו והוכחו אמפירית ברמת מוצר.

תצורות פריסה

המוצר מיועד עקרונית ל:

SDK

Dashboard

SaaS

אופציונלי On-prem

קהל הטבעי הוא:

צוותי AI/ML Engineering

צוותי AI Governance / Risk / Compliance

ספקי LLM

ארגוני Enterprise שמשחררים מערכות AI רגישות

מערכות agents עם צורך בבקרת autonomy וכלי עזר

הצעת הערך

הערך המרכזי של המוצר הוא:

מעבר מ-`evaluation report` ל-`release decision`

הפחתת שחרורים בעייתיים

שיפור `traceability`

הקטנת כאוס בין צוותי `engineering`, `risk` ו-`compliance`

יצירת `evidence exportable` מול הנהלה, לקוחות ורגולציה

בידול

הבידול של LoopGuard AI, כפי שהוגדר, יושב על הצירוף בין:

ייחודיים `metrics`

`decision gates`

`explainability`

`audit trail`

`compliance` של `framing`

מצב בשלות נוכחי

מה שכבר יש:

מסגור מוצרי ברור

ארכיטקטורה עקרונית רצינית

רגילים eval tools רעיוני מול differentiation

מה שחסר כדי להיחשב מוצר מוכח:

formal metric spec

baseline comparisons

repeatable experiments

POC evidence

הנדסי מלא artifact pack

תוצאות מדידות מול use cases אמיתיים

נוסח קצר תפעולי

eval שמאגד אותות, agents-LLMs עבור decision orchestration layer הוא LoopGuard AI וסיכון, מחשב מדדים, מפעיל policy gates, ומחזיר החלטת release מוסברת עם audit trail.

אם תרצה, אנסח עכשיו את זה כ-Product One-Pager, או לחלופין ככרטיס מוצר טכני בטבלה.

רשום את שם האדם שפיתח את loopguard-ai

Thought for 4s

על בסיס המידע שהוגדר כאן, האדם שפיתח את LoopGuard AI הוא בני דונביץ'.