

# RESUMEN EJECUTIVO

## Motor de Inferencia Conversacional

### Proyecto MK

Versión ejecutiva-técnica con capítulos, reglas, métricas y pseudocódigo (E18)

Cliente campo	Monitor ejecutivo	Event Bus RT	E18 Reducer	Storage/Auditoría
---------------	-------------------	--------------	-------------	-------------------

Nota: PDF generado a partir del resumen y el documento E18 provistos en conversación.

Tabla de contenido

Tabla de contenido 2

1. Propósito estratégico 4

2. Arquitectura lógica (visión alto nivel) 4

3. Componentes clave del motor 4

    A. Captura & STT . . . . . 4

    B. Bus de eventos en tiempo real . . . . . 4

    C. Inferencia conversacional . . . . . 4

    D. Cuestionario dinámico . . . . . 5

    E. Monitor ejecutivo . . . . . 5

    F. Persistencia . . . . . 5

4. Motor de inferencia: reglas y métricas 5

    Reglas base . . . . . 5

    Métricas y KPIs (monitor) . . . . . 5

5. Algoritmo 18 (E18): expansión + construcción de sistemas 6

    5.1 Qué hace . . . . . 6

    5.2 Inputs/Outputs . . . . . 6

    5.3 Cuestionario como grafo . . . . . 6

    5.4 Pseudocódigo completo (E18\_TICK) . . . . . 6

6. Esquemas de eventos y contrato de datos 7

    6.1 Envelope estándar (event bus) . . . . . 7

    6.2 SessionState mínimo (persistencia) . . . . . 7

7. Control de expansión y priorización 8

    Caps recomendados . . . . . 8

Score de prioridad (idea) . . . . . 8

8. Construcción de sistemas (System Builder Agent) 8

Ejemplos . . . . . 8

Pseudocódigo (builder) . . . . . 8

9. Endpoints sugeridos (base /var/www/html) 9

RT . . . . . 9

Sesiones . . . . . 9

Ingesta . . . . . 9

Cuestionario/Respuestas . . . . . 9

10. Latencia y presupuestos RT 9

Objetivo . . . . . 9

Budget . . . . . 9

Regla . . . . . 10

11. Robustez y auditoría 10

Auditoría por evento . . . . . 10

Control de cambios . . . . . 10

12. Pruebas mínimas de aceptación 10

Prueba 1 - Expansión . . . . . 10

Prueba 2 - Contradicción . . . . . 10

Prueba 3 - Builder . . . . . 10

13. Referencias 10

## 1. Propósito estratégico

Construir un motor de inferencia conversacional en tiempo real para entrevistas de crédito MK que:

- Transcriba audio continuo con latencia baja.
- Detecte, genere y ajuste preguntas dinámicamente durante la entrevista.
- Infiera respuestas estructuradas con confidence score.
- Alimente en paralelo un monitor ejecutivo y un cliente de campo.
- Genere evidencia auditable (audio, texto, eventos, métricas).

## 2. Arquitectura lógica (visión alto nivel)

Pipeline único -> múltiples receptores

```
AudioStream
-> STT streaming
  -> Event Bus (RT)
    -> Inferencia IA (Q/A, scoring)
    -> Motor de Cuestionario Dinámico
    -> Monitor Ejecutivo
    -> Almacenamiento (crudo + estructurado)
```

- Núcleo único de streaming evita duplicación.
- Receptores desacoplados (monitor, cliente, etiquetadores).
- Ejecución asíncrona y paralela (multi-agente).

## 3. Componentes clave del motor

### A. Captura & STT

- Micrófono único, streaming continuo.
- Buffer offline-first.
- Latencia objetivo: < 200-300 ms.

### B. Bus de eventos en tiempo real

- Estados: listening | sending | receiving | offline | synced.
- Emite eventos de texto, audio, métricas y estados.

### C. Inferencia conversacional

- Matching heurístico + IA.
- Extracción de respuestas por intención.

- Generación de nuevas preguntas si el contexto lo exige.
- Scoring por respuesta [0-1].

#### **D. Cuestionario dinámico**

- No es fijo.
- Se expande/contrae según señales de la conversación.
- Permite overwrite y retroalimentación en vivo.

#### **E. Monitor ejecutivo**

- Vista RT del estado de la entrevista.
- Señales de calidad (silencios, ambigüedad, gaps).
- KPIs visibles (latencia, completitud, confianza).

#### **F. Persistencia**

- Audio crudo.
- Transcripción incremental.
- JSON/YAML estructurado por sesión.
- Listo para auditoría y generación documental.

### **4. Motor de inferencia: reglas y métricas**

#### **Reglas base**

- Event-sourcing primero: si no es evento, no pasó.
- Pipeline único / múltiples receptores.
- Inferencia paralela desacoplada (multi-agente).
- Cuestionario como grafo (no lista).
- Expansión controlada (caps + policy).
- Degradación elegante (IA -> heurística, red -> offline).

#### **Métricas y KPIs (monitor)**

- Latencia STT p95 (ms).
- Latencia tick de inferencia p95 (ms).
- Respuestas con confidence  $\geq 0.72$  (%).

- Preguntas nuevas por sesión.
- Contradicciones detectadas/resueltas.
- Crash/offline rate.
- Tiempo total de entrevista vs baseline.

## 5. Algoritmo 18 (E18): expansión + construcción de sistemas

### 5.1 Qué hace

- Escucha (audio -> STT), comprende, expande (preguntas dinámicas), orquesta (tareas y agentes), audita y construye sistemas (módulos/endpoints/plantillas) a partir de señales del caso.

### 5.2 Inputs/Outputs

- Input: audio\_chunk, transcript\_delta, session\_context, policy\_pack, tool\_catalog.
- Output: event\_stream, answer\_updates, question\_ops, task\_ops, audit\_trail.

### 5.3 Cuestionario como grafo

- Nodos: Q/A/R/T. Ops mínimas: question.add/update/retire/reorder, answer.upsert, rule.attach, task.enqueue.

### 5.4 Pseudocódigo completo (E18\_TICK)

```
function E18_TICK(input):
    # input: transcript_delta | timer_tick | task_result | manual_override

    S = load_session_state(input.session_id)

    emit(SESSION_STATE if changed)

    if input.kind == TRANSCRIPT_DELTA:
        S.buffer.append(input.text)
        S.last_text_ts = now()

    # 1) Feature extraction (rápido, determinista)
    feats = extract_features(S.buffer, S.answers, S.questions)
    # feats: entities, amounts, dates, negations, contradictions hints, silence ratio

    # 2) Parallel inference (agentes)
    jobs = []
    jobs += spawn(agent_extract_answers, feats, S.policy)
    jobs += spawn(agent_detect_gaps, feats, S.question_graph, S.policy)
    jobs += spawn(agent_consistency_check, feats, S.answers)
    jobs += spawn(agent_system_builder, feats, S.tool_catalog, S.policy)

    results = join_with_deadline(jobs, ms=120) # presupuesto RT

    # 3) Deterministic reducer
    proposed_answers = merge_answers(results.answers, tie_break="confidence+recency+evidence")
```

```

proposed_qops    = merge_question_ops(results.qops, policy=S.policy, cap=S.expansion_caps)
proposed_tasks   = merge_tasks(results.tasks, policy=S.policy)

# 4) Apply with audit
for a in proposed_answers:
    if validate(a):
        S.answers.upsert(a)
        emit(ANSWER_UPSERT(a))
    else:
        emit(AUDIT_NOTE("answer_rejected", reason))

for op in proposed_qops:
    if policy_allows(op) and not duplicate(op):
        S.question_graph.apply(op)
        emit(QUESTION_OP(op))

for t in proposed_tasks:
    if not already_done(t) and policy_allows(t):
        enqueue(t)
        emit(TASK_OP("enqueue", t))

# 5) Update health + metrics
emit(METRIC("tick_ms", elapsed_ms))
persist(S)
return

```

## 6. Esquemas de eventos y contrato de datos

### 6.1 Envelope estándar (event bus)

```

{
  "v": 1,
  "ts": 1730000000000,
  "session_id": "mk_2025_12_21_0009",
  "seq": 184,
  "kind": "TRANSCRIPT_DELTA",
  "source": "stt",
  "payload": {},
  "trace": {
    "span": "infer:agentA",
    "parent": "stt:chunk:991"
  }
}

```

### 6.2 SessionState mínimo (persistencia)

```

{
  "session_id": "mk_2025_12_21_0009",
  "advisor_id": "A-13",
  "state": "LISTENING",
  "last_update": 1730000000000,
  "buffer": {
    "text_roll": "...",
    "audio_refs": ["chunk_991.webm", "chunk_992.webm"]
  },
  "answers": {
    "income_monthly": {"value": 18000, "confidence": 0.83, "evidence": ["t:seq178..182"]},
    "dependents": {"value": 3, "confidence": 0.76, "evidence": ["t:seq160..161"]}
  },
}

```

```

"question_graph": {"nodes": [], "edges": []},
"tasks": {"queue": [], "done": []},
"metrics": {"stt_ms_p95": 240, "tick_ms_p95": 110},
"versions": {"policy_pack": "mk_policy_2025_12", "template": "mk_base_v3", "engine": "E18_v1"}
}

```

## 7. Control de expansión y priorización

### Caps recomendados

- max\_new\_questions\_per\_min = 6
- max\_total\_questions = template\_base + 25%
- min\_confidence\_to\_autofill = 0.72
- min\_confidence\_to\_suggest = 0.55
- contradiction\_threshold = 0.30
- repeat\_guard\_window = 90s

### Score de prioridad (idea)

- $\text{score}(q) = w_{\text{gap}} * \text{is\_missing}(q) + w_{\text{risk}} * \text{risk\_weight}(q) + w_{\text{dependency}} * \text{unmet\_dependencies}(q) + w_{\text{time}} * \text{time\_pressure}(q) - w_{\text{annoy}} * \text{asked\_recently}(q)$

## 8. Construcción de sistemas (System Builder Agent)

Genera tareas, módulos y plantillas a partir del contexto del caso (reglas + catálogo de herramientas).

### Ejemplos

- Ingreso + gastos -> task: calc\_cashflow().
- Documentos faltantes -> task: doc\_request\_list.
- Contradicción de domicilio -> follow-up + verificación.
- Futuro visión -> placeholder vision.capture\_id().

### Pseudocódigo (builder)

```

function agent_system_builder(feats, tool_catalog, policy):
    tasks = []
    qops = []

    if feats.has_income and feats.has_expenses:
        tasks += Task("calc_cashflow", inputs=["income", "expenses"], outputs=["net_monthly"])

    if feats.missing("address") and policy.can_ask("address"):
        qops += QuestionAdd(id="addr_current", type="address", priority="high",
                           prompt="Confírmame tu domicilio actual (calle, num, col, mun).")

```



```

if feats.contradiction("employment_status"):
    qops += QuestionAdd(id="job_status_clarify", type="enum",
                        prompt="Me dijiste dos cosas distintas. ¿Actualmente trabajas? Sí/No/Temp")

if tool_catalog.has("docgen") and feats.case_stage >= 2:
    tasks += Task("docgen_pack", inputs=["answers"], outputs=["pdfs", "yaml"])

return {tasks, qops}

```

## 9. Endpoints sugeridos (base /var/www/html)

### RT

- GET /api/rt/stream?session\_id=... -> SSE o WebSocket
- POST /api/rt/event -> inyectar eventos (debug/override)

### Sesiones

- GET /api/sessions?mode=list
- GET /api/sessions?mode=session&id;=...

### Ingesta

- POST /api/ingest/audio -> chunk + meta
- POST /api/ingest/transcript -> delta

### Cuestionario/Respuestas

- POST /api/questions/apply -> ops add/update/retire/reorder
- POST /api/answers/upsert -> corrección manual

## 10. Latencia y presupuestos RT

### Objetivo

- Hablas -> transcripción < 300ms (meta).
- Guía inferida < 500-800ms (mejor esfuerzo).

### Budget

- STT: 150-300ms
- Bus + fanout: 10-30ms
- Agentes: 60-180ms (deadline rígido)

- Reducer + persist: 20-60ms

## Regla

- Si se pasa de deadline: emitir parcial, nunca congelarse.

## 11. Robustez y auditoría

### Auditoría por evento

- Cada ANSWER\_UPSERT guarda valor, confidence, evidencia, versiones (engine/policy/template).
- Replay: event\_log reconstruye el caso completo.

### Control de cambios

- policy\_pack versionado
- template versionado
- engine versionado

## 12. Pruebas mínimas de aceptación

### Prueba 1 - Expansión

- Ingreso sin dependientes -> agrega pregunta dependientes (prioridad alta).

### Prueba 2 - Contradicción

- '2 hijos' y luego '3 hijos' -> follow-up + NEEDS\_CLARIFICATION.

### Prueba 3 - Builder

- Ingreso + gastos -> enqueue(calc\_cashflow) y escribe net\_monthly al terminar.

## 13. Referencias

- Fowler, M. (2005). Event Sourcing.
- Lamport, L. (1998). The Part-Time Parliament. ACM TOCS.
- Fielding, R. (2000). Architectural Styles and the Design of Network-based Software Architectures (REST).

- OpenAI (2022). Whisper: Robust Speech Recognition via Large-Scale Weak Supervision. arXiv:2212.04356.
- Kleppmann, M. (2017). Designing Data-Intensive Applications.