**SST-RiskGPT Vision**

**Análisis visual de riesgos mecánicos y locativos con recomendaciones por Jerarquía de Control**

**Versión:** 1.0  
**Fecha:** 25/09/2025  
**Autor:**

**1. Resumen ejecutivo**

Este documento describe el diseño y la implementación de **SST-RiskGPT Vision**, un prototipo funcional para **detectar riesgos mecánicos y locativos** en imágenes y videos. La solución combina un modelo de detección **YOLOv8** con una **ontología de riesgos** y un **motor de reglas** que traduce detecciones en **recomendaciones priorizadas** conforme a la **Jerarquía de Control** (eliminación, sustitución, ingeniería, administrativos y EPP). Se incluyen interfaz web, API de integración y un módulo de evaluación.

**Resultados clave:**

* Normalización de escenarios de riesgo, recomendaciones y bases normativas en un único archivo (risk\_ontology.yaml).
* Presentación textual de hallazgos y controles en lenguaje claro y consistente.
* Procedimiento de evaluación cuantitativa (Precision/Recall/F1 micro y Kappa) contra etiquetas de experto.

**2. Alcance y objetivos**

**2.1 Alcance**

Primera iteración enfocada en tres familias de riesgo:

1. **Atrapamiento/Atropellamiento por equipos móviles.**
2. **Contacto con partes móviles / maquinaria.**
3. **Caídas al mismo nivel por obstáculos (orden y aseo).**

La ontología es **extensible** (nuevos riesgos, controles, normas).

**2.2 Objetivos (trazabilidad)**

* **(2.4) Normalización:** Definir una **ontología** de riesgos con *disparadores* (clases detectadas), **recomendaciones** por Jerarquía de Control y **normas base** (Decreto 1072/2015, Resolución 0312/2019, ISO 45001:2018).  
  **Entregable:** risk\_ontology.yaml.
* **(2.5) Presentación textual:** Implementar una **capa de salida** que explique hallazgos y controles de forma clara, con terminología SST y referencias normativas.  
  **Entregable:** chat\_layer.py integrado en UI/API.
* **(2.6) Evaluación:** Establecer un **procedimiento de evaluación** con métricas objetivas frente a etiquetas de experto y dataset de prueba.  
  **Entregable:** evaluator.py y guía de uso.

**3. Requisitos y entorno**

* **Python:** 3.10 o superior
* **Dependencias principales:** Ultralytics/YOLOv8, PyTorch, OpenCV, Gradio (UI), FastAPI (API), scikit-learn (métricas).
* **Sistema operativo:** Windows 10/11 (validado).
* **Hardware:** CPU; opcional **GPU NVIDIA** para acelerar entrenamiento/inferencia.
* **Video:** Requiere **FFmpeg** instalado.

**4. Arquitectura de la solución**

**4.1 Componentes**

* **Detector (detector.py):** carga el modelo YOLO, ejecuta inferencia y devuelve clases presentes.
* **Ontología (risk\_ontology.yaml):** catálogo de riesgos, disparadores y controles (Jerarquía).
* **Motor de reglas (rules\_engine.py):** evalúa all\_of/any\_of y produce riesgos + recomendaciones.
* **Capa de presentación (chat\_layer.py):** genera texto claro para usuario/acta.
* **UI imágenes (ui\_gradio.py) y UI video (ui\_gradio\_video.py):** prueba interactiva.
* **API (app\_fastapi.py):** endpoints POST /analyze y POST /analyze-chat.
* **Entrenamiento (train\_yolo.py):** fine-tuning del detector con dataset YOLO.
* **Evaluación (evaluator.py):** cálculo de Precision/Recall/F1 micro y Kappa.

**4.2 Flujo de extremo a extremo**

**Imagen/Video → (YOLOv8) → Clases detectadas → (Reglas) → Riesgos inferidos → (Capa textual) → Recomendaciones + Normas → UI/API.**

**5. Ontología de riesgos (objetivo 2.4)**

La ontología consolida:

* **Normas base:** Decreto 1072/2015 (SG-SST), Resolución 0312/2019 (estándares mínimos), ISO 45001:2018.
* **Riesgos:** nombre, tipo, disparadores (combinaciones de clases), recomendaciones por Jerarquía.
* **Ejemplo (extracto):**

normas:

decreto\_1072\_2015: "Decreto 1072 de 2015 - SG-SST (Colombia)"

res\_0312\_2019: "Resolución 0312 de 2019 - Estándares mínimos SG-SST"

iso\_45001\_2018: "ISO 45001:2018 - Sistemas de gestión de SST"

riesgos:

atrapamiento\_atropellamiento:

tipo: "mecanico"

disparadores:

- all\_of: [person]

any\_of: [forklift, truck, car, excavator, bus]

recomendaciones:

ingenieria:

- "Barreras físicas, demarcación de rutas y espejos en cruces."

epp:

- "Chaleco alta visibilidad, casco, botas con puntera."

normas: [decreto\_1072\_2015, res\_0312\_2019, iso\_45001\_2018]

La ontología es editable por el equipo SST.

**6. Datos y entrenamiento del detector**

**6.1 Formato de dataset**

Estructura YOLO:

D:/datasets/sst/

├─ images/{train,val,test}

└─ labels/{train,val,test}

Cada imagen X.jpg posee un “espejo” X.txt con líneas:  
class\_id x\_center y\_center width height (valores 0..1).

**Clases base (14):** person, forklift, truck, car, excavator, bus, conveyor, machine, saw, press, pallet, cable, spill, toolbox.

**6.2 Generación sintética (opcional, para pruebas)**

El script gen\_synthetic\_sst.py crea un dataset sintético para validar el pipeline:

python gen\_synthetic\_sst.py --root D:/datasets/sst --train 600 --val 200 --test 100 --imgw 1280 --imgh 720

**6.3 Entrenamiento**

# entrenamiento estándar (CPU)

python train\_yolo.py --data D:/datasets/sst/dataset.yaml --model yolov8s.pt --epochs 80 --imgsz 640

**Buenas prácticas:**

* Usar imágenes reales del entorno.
* Balancear clases difíciles (p. ej., cable, spill).
* Iterar incorporando falsos positivos/negativos del terreno.

**7. Inferencia y presentación (objetivo 2.5)**

**7.1 UI imágenes**

python ui\_gradio.py

Navegar a: http://127.0.0.1:7860

* Subir imagen.
* (Opcional) Seleccionar best.pt.
* Se muestran detecciones, riesgos, jerarquía de control y normas aplicables.
* La salida textual se genera desde chat\_layer.py.

**7.2 UI video**

python ui\_gradio\_video.py

http://127.0.0.1:7861

* Cargar clip corto (MP4).
* Exporta video anotado y listado de riesgos por cuadro/segmento.

**7.3 API (integración)**

uvicorn app\_fastapi:app --host 127.0.0.1 --port 8000

http://127.0.0.1:8000/docs

* **POST /analyze:** imagen → JSON con classes\_present, detections, risks, recommendations.
* **POST /analyze-chat:** imagen → lo anterior + **explicación en texto**.

**8. Evaluación de desempeño (objetivo 2.6)**

**8.1 Protocolo**

1. Preparar **ground truth** con etiquetas de experto (por imagen, riesgos esperados).
2. Ejecutar evaluator.py y calcular: **Precision/Recall/F1 (micro)** y **Cohen’s Kappa**.
3. Analizar discrepancias → retroalimentar dataset y reglas.

**Ejemplo de uso:**

python evaluator.py --csv D:/datasets/sst/ground\_truth.csv --model runs/train/sst\_yolo/weights/best.pt

**Estructura ground\_truth.csv:**

image\_path,risks

D:/datasets/sst/images/test/img001.jpg,atrapamiento\_atropellamiento

D:/datasets/sst/images/test/img002.jpg,contacto\_partes\_moviles;caidas\_mismo\_nivel\_obstaculos

**8.2 Criterios de aceptación (sugeridos)**

* **F1 micro ≥ [0.70]** en el conjunto de prueba.
* **Kappa ≥ [0.60]** frente al criterio del experto.  
  *(Sustituir umbrales por los acordados con el cliente).*

**9. Cumplimiento y referencias normativas**

* **Decreto 1072 de 2015 (Colombia):** Sistema de Gestión de Seguridad y Salud en el Trabajo (SG-SST).
* **Resolución 0312 de 2019:** Estándares mínimos SG-SST.
* **ISO 45001:2018:** Sistemas de gestión de SST.

**Nota:** Las recomendaciones priorizan controles de ingeniería y administrativos; el EPP se plantea como última barrera. La adopción final debe ser validada por el **responsable SST** del sitio.

**10. Resultados de prueba (registro)**

**Instrucción:** incorporar capturas de la UI y respuestas del endpoint.

**10.1 Evidencias UI (imágenes)**

* 

- Caídas al mismo nivel por obstáculos / orden y aseo



Sin inferencias de riesgo.

**11. Operación**

* **Lanzar UI (imágenes):** python ui\_gradio.py → http://127.0.0.1:7860
* **Lanzar UI (video):** python ui\_gradio\_video.py → http://127.0.0.1:7861
* **API:** uvicorn app\_fastapi:app --host 127.0.0.1 --port 8000 → /docs
* **Modelo entrenado:** seleccionar runs/train/sst\_yolo4/weights/best.pt (UI) o fijar ruta en app\_fastapi.py.

**12. Limitaciones y riesgos del sistema**

* **Dominio visual:** el desempeño depende de la similitud entre dataset y entorno real (iluminación, ángulos, oclusión).
* **Clases finas:** elementos delgados (p. ej., **cables**) requieren más datos y limpieza de etiquetas.
* **No reemplaza inspección humana:** sirve como **apoyo**; la decisión final es del responsable SST.

**13. Plan de mejora continua**

* Ampliar **clases** (EPP, señalización, zonas prohibidas).
* Recolectar **datos in situ**, reentrenar periódicamente.
* Umbrales por **clase** y lógica espacial (distancias persona–equipo).
* Generar **reportes automáticos** (PDF/Excel) y evidencias para auditoría.

**14. Conclusiones**

SST-RiskGPT Vision cumple los objetivos propuestos: (2.4) normalización del conocimiento de riesgo, (2.5) presentación textual operativa y (2.6) evaluación cuantitativa. La arquitectura es modular, editable por el equipo SST y apta para evolución gradual hacia producción.

**Anexo A — Comandos útiles**

Activar entorno (Windows)

D:/SST-RiskGPT Vision/.venv/Scripts/Activate.ps1

Generar dataset sintético (opcional)

python gen\_synthetic\_sst.py --root D:/datasets/sst --train 600 --val 200 --test 100 --imgw 1280 --imgh 720

Entrenar (CPU)

python train\_yolo.py --data D:/datasets/sst/dataset.yaml --model yolov8s.pt --epochs 80 --imgsz 640

UI imágenes

python ui\_gradio.py # http://127.0.0.1:7860

UI video

python ui\_gradio\_video.py # http://127.0.0.1:7861

API

uvicorn app\_fastapi:app --host 127.0.0.1 --port 8000 # http://127.0.0.1:8000/docs

**Anexo B — Estructura de proyecto (resumen)**

├─ requirements.txt

├─ risk\_ontology.yaml

├─ detector.py

├─ rules\_engine.py

├─ chat\_layer.py

├─ app\_fastapi.py

├─ ui\_gradio.py

├─ video\_analyzer.py

├─ ui\_gradio\_video.py

├─ train\_yolo.py

├─ evaluator.py

└─ gen\_synthetic\_sst.py

**Anexo C — Resolución de problemas (rápido)**

* **“No images found …/images/train”**  
  Verificar que existan imágenes en images/train y sus .txt en labels/train.
* **Error Gradio/localhost**  
  Abrir http://127.0.0.1:7860 y lanzar con server\_name="127.0.0.1".
* **Lento en CPU**  
  Probar yolov8n.pt o menos épocas; usar GPU si disponible.