**SST-RiskGPT Vision**

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

**Versión:** 1.0  
**Fecha:** 25/09/2025  
**Autor:** Michel Andrea Tovar

**1. Resumen ejecutivo**

**SST-RiskGPT Vision es un prototipo funcional para detectar riesgos mecánicos y locativos en imágenes y video y emitir recomendaciones priorizadas según la Jerarquía de Control (eliminación, sustitución, ingeniería, administrativos y EPP). La solución combina detección de objetos (YOLOv8), una ontología de riesgos editable y un motor de reglas que traduce detecciones en riesgos y controles. Incluye UI web, API y un módulo de evaluación con métricas objetivas frente a etiquetas de experto.**

**Resultados clave:**

**- Normalización del conocimiento de riesgo en risk\_ontology.yaml.**

**- Presentación textual coherente y accionable (capa “chat”).**

**- Evaluación reproducible (Precision/Recall/F1 micro y Kappa).**

**- Entrega lista para uso local y compartición (repo + models/best.pt).**

**2. Alcance y objetivos**

**Alcance: Primera iteración centrada en tres familias de riesgo: (1) Atrapamiento/Atropellamiento por equipos móviles; (2) Contacto con partes móviles / maquinaria; (3) Caídas al mismo nivel por obstáculos.**

**Objetivos y trazabilidad:**

**- O1 – Normalización (Ontología): diseñar un modelo declarativo con disparadores (clases detectadas), recomendaciones por Jerarquía de Control y normas base (Decreto 1072/2015, Res. 0312/2019, ISO 45001:2018). Entregable: risk\_ontology.yaml.**

**- O2 – Presentación textual: generar una salida clara que explique hallazgos, controles y referencias normativas. Entregable: chat\_layer.py + integración en UI/API.**

**- O3 – Evaluación: establecer un protocolo y métricas frente a etiquetas de experto. Entregable: evaluator.py + guía de uso.**

**3. Requisitos y entorno**

**Sistema: Windows 10/11 validado.**

**Python: 3.10–3.11 recomendado.**

**Dependencias: Ultralytics/YOLOv8, PyTorch, OpenCV, Gradio (UI), FastAPI (API), scikit-learn (métricas).**

**Video: requiere FFmpeg instalado.**

**Hardware: CPU funcional; GPU NVIDIA acelera entrenamiento/inferencia.**

**4. Arquitectura de la solución**

**Componentes:**

**- Detección (detector.py): carga pesos YOLO, infiere y expone clases presentes.**

**- Ontología (risk\_ontology.yaml): catálogo de riesgos, disparadores y controles por Jerarquía.**

**- Motor de reglas (rules\_engine.py): evalúa all\_of/any\_of sobre clases detectadas y produce riesgos + recomendaciones.**

**- Capa textual (chat\_layer.py): redacción consistente y breve para salida y actas.**

**- UI imágenes (ui\_gradio.py) y UI video (ui\_gradio\_video.py): prueba interactiva.**

**- API (app\_fastapi.py): endpoints /analyze y /analyze-chat.**

**- Entrenamiento (train\_yolo.py): fine-tuning a partir de pesos base YOLO.**

**- Evaluación (evaluator.py): métricas objetivas frente a ground truth.**

**Flujo extremo a extremo:**

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

**5. Ontología de riesgos (O1)**

**La ontología consolida:**

**- Normas base: Decreto 1072/2015 (SG-SST), Resolución 0312/2019 (estándares mínimos), ISO 45001:2018.**

**- Riesgos: identificador, nombre, tipo, disparadores (combinaciones de clases), recomendaciones por Jerarquía de Control y normas aplicables.**

**Extracto ilustrativo (YAML):**

**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."**

**administrativos:**

**- "Plan de tráfico interno, permisos y verificación pre-operacional."**

**epp:**

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

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

**6. Datos y entrenamiento**

**Estructura del dataset (YOLO):**

**datasets/sst/**

**├─ images/{train,val,test}**

**└─ labels/{train,val,test}**

**Cada imagen X.jpg tiene un X.txt con líneas: class\_id x\_center y\_center width height (valores 0..1).**

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

**Generación sintética (opcional):**

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

**Entrenamiento:**

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

**Buenas prácticas: incluir imágenes reales del sitio; balancear clases difíciles; iterar con falsos positivos/negativos de inspecciones reales.**

**7. Inferencia y presentación (O2)**

**UI de imágenes:**

**python ui\_gradio.py → http://127.0.0.1:7860**

**- Campo “Ruta a modelo YOLO entrenado (.pt)” preconfigurado con models/best.pt.**

**- Ajustes sugeridos: Confianza 0.25–0.35; IoU 0.60.**

**UI de video:**

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

**API (integración):**

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

**- POST /analyze → JSON con detecciones, riesgos, recomendaciones.**

**- POST /analyze-chat → incluye texto explicativo consolidado.**

**8. Evaluación de desempeño (O3)**

**Protocolo: comparar salida de riesgos con ground truth de experto (a nivel de imagen), con métricas F1 micro y Kappa, analizar discrepancias y retroalimentar dataset y reglas.**

**Ejemplo de uso:**

**python evaluator.py --csv datasets/sst/ground\_truth.csv --model models/best.pt**

**9. Cumplimiento normativo**

**Decreto 1072/2015 (Colombia) – SG-SST; Resolución 0312/2019 – Estándares mínimos; ISO 45001:2018 – SG SST. La herramienta prioriza controles de ingeniería y administrativos; el EPP se considera última barrera. La adopción final se valida con el responsable SST del sitio.**

**10. Entrega, compartición y reproducibilidad**

**.gitignore relevante:**

**runs/**

**\*.pt**

**!models/**

**!models/\*.pt**

**Clonar y ejecutar**

**git lfs install**

**git clone https://github.com/<TU\_USUARIO>/SST-RiskGPT-Vision.git**

**cd SST-RiskGPT-Vision**

**python -m venv .venv**

**.\.venv\Scripts\Activate.ps1**

**pip install --upgrade pip**

**pip install -r requirements.txt**

**python ui\_gradio.py**

**11. Evidencias**

****

**Riesgo de caída.**

**12. Limitaciones y riesgos del sistema**

**Desempeño ligado a similitud entre dataset y entorno real; clases finas (cables/derrames) requieren más datos; apoyo a inspección, no reemplaza juicio profesional.**

**13. Plan de mejora continua**

**Ampliar clases (EPP, señalización), recolectar datos in situ y re-entrenar, umbrales por clase y relaciones espaciales, reportes PDF/Excel y consolidación para auditoría.**

**14. Conclusiones**

**Se cumplen los objetivos: O1 normaliza conocimiento de riesgo; O2 presenta resultados claros y accionables; O3 define evaluación reproducible. Arquitectura modular, editable por el equipo SST y apta para evolución hacia producción.**

**Anexo A — Comandos útiles**

**Activar entorno (Windows):**

**.\.venv\Scripts\Activate.ps1**

**Generar dataset sintético:**

**python gen\_synthetic\_sst.py --root datasets/sst --train 600 --val 200 --test 100**

**Entrenar (CPU):**

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

**UI imágenes / video:**

**python ui\_gradio.py (http://127.0.0.1:7860)**

**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 del proyecto**

**SST-RiskGPT Vision/**

**├─ app\_fastapi.py**

**├─ ui\_gradio.py**

**├─ ui\_gradio\_video.py**

**├─ video\_analyzer.py**

**├─ detector.py**

**├─ rules\_engine.py**

**├─ chat\_layer.py**

**├─ risk\_ontology.yaml**

**├─ train\_yolo.py**

**├─ evaluator.py**

**├─ gen\_synthetic\_sst.py**

**├─ models/**

**│ └─ best.pt**

**├─ datasets/**

**│ ├─ sst\_sample/**

**│ └─ .gitkeep**

**├─ scripts/**

**│ ├─ configure\_dataset.py**

**│ └─ (opcional) prepare\_weights.py**

**├─ requirements.txt**

**└─ .gitignore**

**Anexo C — Resolución de problemas**

**“No images found …/images/train” → verificar imágenes y etiquetas.**

**Gradio/localhost → abrir http://127.0.0.1:7860; si hay restricción de red, share=True.**

**Lento en CPU → usar yolov8n.pt, menos épocas, o GPU.**

**best.pt no cargado → confirmar models/best.pt o pegar ruta válida; con LFS, git lfs pull.**