Application of Ramtera LVR with Google Cloud

Prepared for Google Cloud Partner Solution Qualification

Ramtera Design Automation Inc. (Delaware, USA)

https://ramtera.com

Date: October 29, 2025

Ramtera Inc

December 20, 2025


LVR
Layout Verification
Reporter


RAMTERA INC
FAST AND ACCURATE EDA TOOLS

Contents

1 LVR Application with GCloud

LVR has been containerized and executed on Google Cloud Compute Engine (GCE) to validate trillion-polygon physical verification scalability. A startup script (run_lvr_on_cloud.sh) provisions Ubuntu 24.04 VMs, installs Qt5/TCL/libstdc++ dependencies, uploads design data to Cloud Storage, executes LVR_batch commands remotely, and stores verification reports back in a GCS bucket. The workflow is entirely driven by the Google Cloud SDK (gcloud, gsutil) from a macOS workstation. This application demonstrates elastic compute, data persistence, and reproducible verification runs on Google Cloud infrastructure.

2 LVR Description

Ramtera LVR (Layout Verification Reporter) is an EDA (Electronic Design Automation) platform written primarily in C++17 with Qt5 for GUI and Boost libraries for computational geometry. It performs DRC/LVS/ERC/XOR/Density verification for IC layouts. The tool runs as a multithreaded native executable, with TCL scripting for automation, YAML/JSON configuration for rule decks, and SQLite for intermediate data caching. LVR executes on Linux (Ubuntu 22.04 / 24.04 LTS) and macOS, using its own in-memory database for polygon and net connectivity. The product integrates with standard GDSII/OASIS layout formats and SPICE/Verilog netlists.

3 Shell script 00_config.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Shared configuration file used by all scripts
#  Author: Ramtera Design Automation (Ramtera Inc.)
#  Description: Defines environment variables and default settings
# ============================================================
set -euo pipefail

# ====== Common Config (override via env before sourcing) ======
```

```
PROJECT_ID="${PROJECT_ID:-$(gcloud config get-value project 2>/dev/null || true)}"
REGION="${REGION:-$(gcloud config get-value compute/region 2>/dev/null || echo us-central1)}"
ZONE="${ZONE:-${ZONE:-$(gcloud config get-value compute/zone 2>/dev/null || echo us-central1-a)}}"
INSTANCE_NAME="${INSTANCE_NAME:-lvr-node-1}"
MACHINE_TYPE="${MACHINE_TYPE:-n2-highmem-16}"
BOOT_DISK_SIZE_GB="${BOOT_DISK_SIZE_GB:-10}"
IMAGE_FAMILY="${IMAGE_FAMILY:-ubuntu-2404-lts}"
IMAGE_PROJECT="${IMAGE_PROJECT:-ubuntu-os-cloud}"

# Default bucket name; will be overridden by .ramtera_state.env if present
BUCKET_NAME="${BUCKET_NAME:-ramtera-lvr-data-$(date +%Y%m%d-%H%M%S)}"

OUT_LOCAL="${OUT_LOCAL:-./out_local}"
WORKDIR="/opt/ramtera/lvr_run"

# Load sticky state if present (keeps same bucket across runs)
if [[ -f ".ramtera_state.env" ]]; then
  # shellcheck disable=SC1091
  source ".ramtera_state.env"
fi

[[ -n "${PROJECT_ID}" ]] || { echo "ERROR: PROJECT_ID not set and no gcloud default project."; exit 1; }
mkdir -p "${OUT_LOCAL}"
```

4 Shell script 01_setup_auth.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Step 1: Setup login, authentication, billing, and VM creation
#  Author: Ramtera Design Automation (Ramtera Inc.)
# ============================================================
set -euo pipefail
source "$(dirname "$0")/00_config.sh"

# Tools present?
for b in gcloud gsutil; do command -v "$b" >/dev/null || { echo "ERROR: missing $b"; exit 1; }; done

echo "Project: ${PROJECT_ID}"
echo "Region/Zone: ${REGION}/${ZONE}"
echo "VM: ${INSTANCE_NAME}  Machine: ${MACHINE_TYPE}"
echo "Bucket: gs://${BUCKET_NAME}"
echo

# Make sure you're logged in (no-op if already logged in)
gcloud auth login --brief || true
gcloud config set project "${PROJECT_ID}"

# Fail early if billing not linked
if ! gcloud beta billing projects describe "${PROJECT_ID}" --format="value(billingEnabled)" 2>/dev/null | grep -qi true; then
  echo "ERROR: Billing not enabled for ${PROJECT_ID}."
  echo "Fix: gcloud beta billing accounts list"
  echo "     gcloud beta billing projects link ${PROJECT_ID} --billing-account=ACCOUNT_ID"
  exit 1
fi

# Enable required APIs
gcloud services enable compute.googleapis.com --project "${PROJECT_ID}"
gcloud services enable storage.googleapis.com --project "${PROJECT_ID}"

# Bucket (idempotent)
if gsutil ls -b "gs://${BUCKET_NAME}" >/dev/null 2>&1; then
  echo "Bucket exists: gs://${BUCKET_NAME}"
else
  gsutil mb -p "${PROJECT_ID}" -l "${REGION}" "gs://${BUCKET_NAME}"
fi

# Persist chosen bucket so other scripts reuse the same one
STATE_FILE=".ramtera_state.env"
grep -v '^BUCKET_NAME=' "${STATE_FILE}" 2>/dev/null > "${STATE_FILE}.tmp" || true
echo "BUCKET_NAME=${BUCKET_NAME}" >> "${STATE_FILE}.tmp"
mv "${STATE_FILE}.tmp" "${STATE_FILE}"
echo "Saved bucket to ${STATE_FILE}"

# Grant the VM's default service account access to the bucket (avoids 403 on VM)
PROJECT_NUMBER="$(gcloud projects describe "${PROJECT_ID}" --format='value(projectNumber)')"
SA="${PROJECT_NUMBER}-compute@developer.gserviceaccount.com"
if ! gsutil iam get "gs://${BUCKET_NAME}" 2>/dev/null | grep -q "${SA}"; then
  echo "Granting objectAdmin on gs://${BUCKET_NAME} to ${SA} ..."
  gsutil iam ch "serviceAccount:${SA}:objectAdmin" "gs://${BUCKET_NAME}"
fi

# Resolve Ubuntu 24.04; fallback to 22.04
IMAGE_NAME="$(gcloud compute images describe-from-family ${IMAGE_FAMILY} --project=${IMAGE_PROJECT} --format='get(name)')" || IMAGE_NAME=""
if [[ -z "${IMAGE_NAME}" ]]; then
```

```
  echo "FALLBACK: trying ubuntu-2204-lts..."
  IMAGE_NAME="$(gcloud compute images describe-from-family ubuntu-2204-lts --project=ubuntu-os-cloud --format='get(name)')"
  [[ -n "${IMAGE_NAME}" ]] || { echo "ERROR: could not resolve Ubuntu image"; exit 1; }
  IMAGE_PROJECT="ubuntu-os-cloud"
fi
echo "Using image: ${IMAGE_NAME}"

# Create or reuse VM
if gcloud compute instances describe "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null 2>&1; then
  echo "Reusing VM ${INSTANCE_NAME}"
else
  gcloud compute instances create "${INSTANCE_NAME}" \
    --project "${PROJECT_ID}" \
    --zone "${ZONE}" \
    --machine-type "${MACHINE_TYPE}" \
    --image "${IMAGE_NAME}" \
    --image-project "${IMAGE_PROJECT}" \
    --boot-disk-size "${BOOT_DISK_SIZE_GB}" \
    --boot-disk-type pd-ssd \
    --scopes "https://www.googleapis.com/auth/cloud-platform"
fi

# Ensure VM is RUNNING
gcloud compute instances start "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true

# --- IAP SSH setup ---
# Create IAP firewall rule (idempotent)
if ! gcloud compute firewall-rules describe allow-iap-ssh --project "${PROJECT_ID}" >/dev/null 2>&1; then
  gcloud compute firewall-rules create allow-iap-ssh \
    --project="${PROJECT_ID}" \
    --network=default \
    --direction=INGRESS --action=ALLOW \
    --rules=tcp:22 \
    --source-ranges=35.235.240.0/20
fi
# Tag instance so rule applies (safe if already present)
gcloud compute instances add-tags "${INSTANCE_NAME}" \
  --project="${PROJECT_ID}" --zone="${ZONE}" \
  --tags=allow-iap-ssh >/dev/null || true

# Helper to test SSH via IAP
try_iap_ssh() {
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    --tunnel-through-iap --command "echo VM up" >/dev/null 2>&1
}

# Try IAP (retries), else fallback to public IP
echo "Waiting for SSH... (via IAP)"
USE_IAP=0
for i in {1..8}; do
  if try_iap_ssh; then USE_IAP=1; break; fi
  echo "SSH (IAP) not ready yet...retry $i/8"; sleep 8
done

if [[ "${USE_IAP}" -eq 0 ]]; then
  echo "Falling back to temporary external IP for SSH..."
  gcloud compute instances add-access-config "${INSTANCE_NAME}" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
  if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
         --command "echo VM up" >/dev/null 2>&1; then
    echo "ERROR: SSH failed even after attaching external IP."
    echo "Debug:"
```

```
    echo "  gcloud compute instances get-serial-port-output ${INSTANCE_NAME} --zone ${ZONE} --project ${PROJECT_ID} --port=1 | tail -n 200"
    exit 1
  fi
  SSH_EXTRAS=()   # public path
else
  SSH_EXTRAS=(--tunnel-through-iap)
fi

# Small wrapper to run SSH commands using the detected path
ssh_run() {
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    "${SSH_EXTRAS[@]}" --command "$1"
}

# --- Ensure outbound egress for apt/gsutil (attach temp IP if needed) ---
NEEDS_EGRESS_FIX=0
if ! ssh_run "curl -fsI http://archive.ubuntu.com/ubuntu/ >/dev/null || ping -c1 -W2 8.8.8.8 >/dev/null"; then
  echo "No outbound internet detected from VM. Attaching temporary external IP..."
  gcloud compute instances add-access-config "${INSTANCE_NAME}" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
  NEEDS_EGRESS_FIX=1
fi

# Mirror fallback (regional mirror can be flaky without NAT)
ssh_run 'sudo sed -i "s|http://us-central1.gce.archive.ubuntu.com/ubuntu|http://archive.ubuntu.com/ubuntu|g" /etc/apt/sources.list || true'

# Install runtimes & ensure GLIBCXX_3.4.32 + crcmod for fast gsutil
ssh_run "set -e; sudo apt-get update -y && sudo apt-get install -y \
   software-properties-common ca-certificates build-essential unzip curl iputils-ping \
   libstdc++6 libgcc-s1 \
   libqt5widgets5 libqt5gui5 libqt5core5a libqt5printsupport5 libqt5svg5 \
   libxkbcommon-x11-0 libxrender1 libx11-xcb1 libxext6 libsm6 libice6 \
   libxcb1 libxcb-render0 libxcb-shape0 libxcb-xfixes0 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 \
   libopengl0 libgl1 libglu1-mesa \
   tcl tcl8.6 tk tk8.6 \
   python3-crcmod"

if ! ssh_run "strings /usr/lib/x86_64-linux-gnu/libstdc++.so.6 | grep -q 'GLIBCXX_3\.4\.32'"; then
  ssh_run "sudo add-apt-repository -y ppa:ubuntu-toolchain-r/test && sudo apt-get update -y && sudo apt-get install -y libstdc++6"
  ssh_run "strings /usr/lib/x86_64-linux-gnu/libstdc++.so.6 | grep -q 'GLIBCXX_3\.4\.32' || (echo 'ERROR: GLIBCXX_3.4.32 missing'; exit 1)"
fi

# Prep workdir
ssh_run "sudo mkdir -p ${WORKDIR}/in ${WORKDIR}/out && sudo chown -R \$USER:\$USER ${WORKDIR}"

# Remove temp external IP used for egress if we attached it above (keep any earlier one used for SSH fallback)
if [[ "${NEEDS_EGRESS_FIX}" -eq 1 ]]; then
  echo "Removing temporary external IP after package installation..."
  gcloud compute instances delete-access-config "${INSTANCE_NAME}" \
    --access-config-name="external-nat" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
fi

# (Optional tip for local speedups)
echo "Tip: for faster local <-> GCS and IAP, on your Mac run:"
echo "  python3 -m pip install --user crcmod numpy"

echo "Setup complete."
```

5 Shell script 02_transfer_to_vm.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Step 2: Transfer local files to GCE VM (GCS fast path)
#  Author: Ramtera Design Automation (Ramtera Inc.)
# ============================================================
set -euo pipefail
source "$(dirname "$0")/00_config.sh"

# Ensure bucket exists (idempotent)
if ! gsutil ls -b "gs://${BUCKET_NAME}" >/dev/null 2>&1; then
  echo "Bucket gs://${BUCKET_NAME} not found | creating it now..."
  gsutil mb -p "${PROJECT_ID}" -l "${REGION}" "gs://${BUCKET_NAME}"
fi

echo "Fast path: syncing local → GCS ..."
# Use checksum mode (-c) to avoid long mtimes/size comparisons; tune threads.
gsutil -o "GSUtil:parallel_process_count=1" \
       -o "GSUtil:parallel_thread_count=32" \
       -m rsync -r -c -x '(^|/)(\.git/|out_local/)|upload\.tgz$' ./ "gs://${BUCKET_NAME}/upload/"

echo "Pulling from GCS on VM ..."
PULL_FAST="mkdir -p ${WORKDIR}/in && gsutil -m cp -r gs://${BUCKET_NAME}/upload/* ${WORKDIR}/in/ || true"
PULL_SAFE="mkdir -p ${WORKDIR}/in && gsutil -m rsync -r gs://${BUCKET_NAME}/upload/ ${WORKDIR}/in/"

# Try fast path (cp), then fallback to rsync for completeness
if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
      --tunnel-through-iap --command "${PULL_FAST}"; then
  echo "IAP cp failed | retrying with rsync..."
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    --tunnel-through-iap --command "${PULL_SAFE}" || \
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    --command "${PULL_SAFE}"
fi

# Make LVR_batch executable if present
if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
      --tunnel-through-iap --command "chmod +x ${WORKDIR}/in/LVR_batch || true"; then
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    --command "chmod +x ${WORKDIR}/in/LVR_batch || true"
fi

echo "Transfer complete."
```

6 Shell script 03_run_lvr.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Step 3: Execute LVR_batch on remote VM
#  Author: Ramtera Design Automation (Ramtera Inc.)
# ============================================================
set -euo pipefail
source "$(dirname "$0")/00_config.sh"

CMD_FILE="${1:-jpeg_sky.cmd}"   # allow overriding the .cmd file via arg

echo "Running LVR_batch on ${CMD_FILE} ..."
RUN_CMD="set -euo pipefail; \
 cd ${WORKDIR}/in && \
 export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:\${LD_LIBRARY_PATH:-}; \
 export QT_QPA_PLATFORM=offscreen; \
 ./LVR_batch ${CMD_FILE}"

if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
      --tunnel-through-iap --command "${RUN_CMD}"; then
  echo "IAP SSH failed | retrying without IAP (public IP path)..."
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    --command "${RUN_CMD}"
fi

echo "Run complete."
```

7 Shell script 04_copy_from_vm.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Step 4: Copy output files from VM → local (SSH tar) and optional GCS upload
# ============================================================
set -euo pipefail
source "$(dirname "$0")/00_config.sh"

MODE="${MODE:-ssh}"   # ssh | egress
UPLOAD_TO_GCS="${UPLOAD_TO_GCS:-true}"

mkdir -p "${OUT_LOCAL}"

if [[ "${MODE}" == "ssh" ]]; then
  echo "Collecting results via SSH (IAP) ..."
  PACK_CMD='set -e;
    cd '"${WORKDIR}"';
    mkdir -p out;  # ensure exists
    # Pack reports/logs & any out/ directory; ignore missing globs
    tar -czf /tmp/lvr_results.tgz \
      --ignore-failed-read \
      in/*.rpt in/*.mrk in/*.log out 2>/dev/null || \
    tar -czf /tmp/lvr_results.tgz in out 2>/dev/null || true'
  # Try IAP first, fallback to public IP path
  if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
        --tunnel-through-iap --command "${PACK_CMD}"; then
    echo "IAP SSH failed | retrying without IAP (public IP)..."
    gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
      --command "${PACK_CMD}"
  fi

  # Copy archive back
  if ! gcloud compute scp --tunnel-through-iap --zone "${ZONE}" \
        "${INSTANCE_NAME}:/tmp/lvr_results.tgz" "${OUT_LOCAL}/"; then
    gcloud compute scp --zone "${ZONE}" \
      "${INSTANCE_NAME}:/tmp/lvr_results.tgz" "${OUT_LOCAL}/"
  fi
  tar -xzf "${OUT_LOCAL}/lvr_results.tgz" -C "${OUT_LOCAL}/" || true
  echo "Local results: ${OUT_LOCAL}"

  if [[ "${UPLOAD_TO_GCS}" == "true" ]]; then
    echo "Uploading results to GCS from local ..."
    gsutil -m rsync -r "${OUT_LOCAL}/" "gs://${BUCKET_NAME}/results/"
    echo "Also in GCS: gs://${BUCKET_NAME}/results/"
  fi

elif [[ "${MODE}" == "egress" ]]; then
  echo "Collecting results via gsutil on VM (requires egress) ..."
  # Ensure VM has egress (attach temp external IP)
  gcloud compute instances add-access-config "${INSTANCE_NAME}" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true

  VM_SYNC_CMD="mkdir -p ${WORKDIR}/out;
    gsutil -m rsync -r ${WORKDIR}/in gs://${BUCKET_NAME}/results/in/;
    gsutil -m rsync -r ${WORKDIR}/out gs://${BUCKET_NAME}/results/out/"
  if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
        --tunnel-through-iap --command "${VM_SYNC_CMD}"; then
    gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
      --command "${VM_SYNC_CMD}"
```

```
  fi

  echo "Syncing GCS → local ..."
  gsutil -m rsync -r "gs://${BUCKET_NAME}/results/" "${OUT_LOCAL}/"
  echo "Local results: ${OUT_LOCAL}"
  echo "Also in GCS:   gs://${BUCKET_NAME}/results/"

  # Remove temp IP
  gcloud compute instances delete-access-config "${INSTANCE_NAME}" \
    --access-config-name="external-nat" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
else
  echo "Unknown MODE: ${MODE} (use ssh|egress)"; exit 2
fi
```

8 Shell script 05_close_vm.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  Step 5: Stop or delete GCE VM (with optional IP cleanup)
#  Author: Ramtera Design Automation (Ramtera Inc.)
# ============================================================
set -euo pipefail
source "$(dirname "$0")/00_config.sh"

ACTION="${1:-stop}"   # stop | delete
REMOVE_EXTERNAL_IP_ON_STOP="${REMOVE_EXTERNAL_IP_ON_STOP:-true}"

maybe_remove_external_ip() {
  if [[ "${REMOVE_EXTERNAL_IP_ON_STOP}" == "true" ]]; then
    # Delete default access-config if present (name is usually 'external-nat')
    if gcloud compute instances describe "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
         --format="get(networkInterfaces[0].accessConfigs[0].name)" 2>/dev/null | grep -q .; then
      echo "Removing temporary external IP (access-config)..."
      gcloud compute instances delete-access-config "${INSTANCE_NAME}" \
        --access-config-name="external-nat" \
        --project "${PROJECT_ID}" --zone "${ZONE}" >/dev/null || true
    fi
  fi
}

case "${ACTION}" in
  stop)
    maybe_remove_external_ip
    echo "Stopping VM ${INSTANCE_NAME} ..."
    gcloud compute instances stop "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
    echo "VM stopped."
    ;;
  delete)
    echo "Deleting VM ${INSTANCE_NAME} ..."
    gcloud compute instances delete "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --quiet || true
    echo "VM deleted."
    ;;
  *)
    echo "Usage: $0 [stop|delete]"
    exit 2
    ;;
esac
```

9 Shell script check_gcs_exists.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  GCS Existence Checker
#  Author: Ramtera Design Automation (Ramtera Inc.)
# ============================================================

BUCKET_NAME="${BUCKET_NAME:-ramtera-lvr-data-$(date +%Y%m%d-%H%M%S)}"

echo "Checking if bucket gs://${BUCKET_NAME} exists..."

if gsutil ls -b "gs://${BUCKET_NAME}" >/dev/null 2>&1; then
  echo " Bucket exists: gs://${BUCKET_NAME}"
else
  echo " Bucket not found: gs://${BUCKET_NAME}"
  echo "Creating bucket..."
  gsutil mb -p "$(gcloud config get-value project)" -l "$(gcloud config get-value compute/region || echo us-central1)" "gs://${BUCKET_NAME}"
  if [[ $? -eq 0 ]]; then
    echo " Bucket created successfully: gs://${BUCKET_NAME}"
  else
    echo " Failed to create bucket: gs://${BUCKET_NAME}"
    exit 1
  fi
fi
```

10 Shell script check_vm_status.sh

```
#!/usr/bin/env bash
# ============================================================
#  Ramtera Inc. | Cloud Automation for LVR
#  GCE VM Status Checker
#  Author: Ramtera Design Automation (Ramtera Inc.)
#  Description:
#    Checks whether a given VM instance exists and is RUNNING.
#    If stopped, it can start the VM automatically.
# ============================================================
set -euo pipefail

# ---- Configuration ----
PROJECT_ID="${PROJECT_ID:-$(gcloud config get-value project 2>/dev/null || true)}"
ZONE="${ZONE:-$(gcloud config get-value compute/zone 2>/dev/null || echo us-central1-a)}"
INSTANCE_NAME="${INSTANCE_NAME:-lvr-node-1}"
AUTO_START="${AUTO_START:-false}"   # change to false if you only want to check

[[ -n "${PROJECT_ID}" ]] || { echo "ERROR: PROJECT_ID not set and no gcloud default project."; exit 1; }

echo "Checking VM status:"
echo "  Project : ${PROJECT_ID}"
echo "  Zone    : ${ZONE}"
echo "  VM Name : ${INSTANCE_NAME}"
echo

# ---- Check if VM exists ----
if ! gcloud compute instances describe "${INSTANCE_NAME}" \
      --project "${PROJECT_ID}" --zone "${ZONE}" >/dev/null 2>&1; then
  echo " VM ${INSTANCE_NAME} does not exist in project ${PROJECT_ID} (${ZONE})."
  exit 1
fi

# ---- Query status ----
STATUS="$(gcloud compute instances describe "${INSTANCE_NAME}" \
          --project "${PROJECT_ID}" --zone "${ZONE}" \
          --format='value(status)')"

echo "Current status: ${STATUS}"

case "${STATUS}" in
  RUNNING)
    echo " VM is already running."
    ;;
  TERMINATED|STOPPED)
    if [[ "${AUTO_START}" == "true" ]]; then
      echo "  VM is stopped | starting now..."
      gcloud compute instances start "${INSTANCE_NAME}" \
        --project "${PROJECT_ID}" --zone "${ZONE}" >/dev/null
      echo " VM started successfully."
    else
      echo "  VM is stopped. AUTO_START=false | leaving it stopped."
    fi
    ;;
  STAGING|PROVISIONING)
    echo " VM is still initializing. Try again shortly."
    ;;
  *)
    echo "  Unknown VM status: ${STATUS}"
    ;;
```

```
esac
```

11 Shell script run_lvr_on_gcloud.sh

```
#!/usr/bin/env bash
set -euo pipefail

# ========= Config (override via env) =========
PROJECT_ID="${PROJECT_ID:-$(gcloud config get-value project 2>/dev/null || true)}"
REGION="${REGION:-$(gcloud config get-value compute/region 2>/dev/null || echo us-central1)}"
ZONE="${ZONE:-$(gcloud config get-value compute/zone 2>/dev/null || echo us-central1-a)}"
INSTANCE_NAME="${INSTANCE_NAME:-lvr-node-1}"
MACHINE_TYPE="${MACHINE_TYPE:-n2-highmem-16}"
BOOT_DISK_SIZE_GB="${BOOT_DISK_SIZE_GB:-10}"
IMAGE_FAMILY="${IMAGE_FAMILY:-ubuntu-2404-lts}"       # Ubuntu 24.04 LTS for GLIBCXX_3.4.32+
IMAGE_PROJECT="${IMAGE_PROJECT:-ubuntu-os-cloud}"
BUCKET_NAME="${BUCKET_NAME:-ramtera-lvr-data-$(date +%Y%m%d-%H%M%S)}"
KEEP_INSTANCE="${KEEP_INSTANCE:-false}"
USE_SPOT="${USE_SPOT:-false}"
OUT_LOCAL="${OUT_LOCAL:-./out_local}"                  # where to fetch results on your Mac
WORKDIR="/opt/ramtera/lvr_run"

# ========= Pre-flight =========
for b in gcloud gsutil; do command -v "$b" >/dev/null || { echo "ERROR: missing $b"; exit 1; }; done
[[ -n "${PROJECT_ID}" ]] || { echo "ERROR: PROJECT_ID not set and no gcloud default project."; exit 1; }
mkdir -p "${OUT_LOCAL}"

echo "Project: ${PROJECT_ID}"
echo "Region/Zone: ${REGION}/${ZONE}"
echo "Image: ${IMAGE_FAMILY} (${IMAGE_PROJECT})"
echo "VM: ${INSTANCE_NAME}  Machine: ${MACHINE_TYPE}"
echo "Bucket: gs://${BUCKET_NAME}"
echo

# Fail early if billing not linked
if ! gcloud beta billing projects describe "${PROJECT_ID}" --format="value(billingEnabled)" 2>/dev/null | grep -qi true; then
  echo "ERROR: Billing not enabled for ${PROJECT_ID}."
  echo "Link billing: gcloud beta billing projects link ${PROJECT_ID} --billing-account=ACCOUNT_ID"
  exit 1
fi

# Enable APIs
gcloud services enable compute.googleapis.com --project "${PROJECT_ID}"
gcloud services enable storage.googleapis.com --project "${PROJECT_ID}"

# Bucket create (idempotent)
if gsutil ls -b "gs://${BUCKET_NAME}" >/dev/null 2>&1; then
  echo "Bucket exists: gs://${BUCKET_NAME}"
else
  gsutil mb -p "${PROJECT_ID}" -l "${REGION}" "gs://${BUCKET_NAME}"
fi

# -------- Resolve Ubuntu 24.04 image by name --------
echo "Resolving Ubuntu 24.04 LTS image..."
IMAGE_NAME="$(gcloud compute images describe-from-family ubuntu-2404-lts \
  --project=ubuntu-os-cloud --format='get(name)')" || IMAGE_NAME=""
if [[ -z "${IMAGE_NAME}" ]]; then
  echo "FALLBACK: ubuntu-24.04 family not found, trying 22.04..."
  IMAGE_NAME="$(gcloud compute images describe-from-family ubuntu-2204-lts \
    --project=ubuntu-os-cloud --format='get(name)')"
  [[ -n "${IMAGE_NAME}" ]] || { echo "ERROR: could not resolve Ubuntu image"; exit 1; }
  export IMAGE_PROJECT="ubuntu-os-cloud"
fi
```

```
echo "Using image: ${IMAGE_NAME}"

# -------- Create VM (or reuse) --------
if gcloud compute instances describe "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null 2>&1; then
  echo "Reusing VM ${INSTANCE_NAME}"
else
  gcloud compute instances create "${INSTANCE_NAME}" \
    --project "${PROJECT_ID}" \
    --zone "${ZONE}" \
    --machine-type "${MACHINE_TYPE}" \
    --image "${IMAGE_NAME}" \
    --image-project "${IMAGE_PROJECT}" \
    --boot-disk-size "${BOOT_DISK_SIZE_GB}" \
    --boot-disk-type pd-ssd \
    --scopes "https://www.googleapis.com/auth/cloud-platform" \
    "${SPOT_FLAGS[@]}"
fi

echo "Waiting for SSH..."
echo "Waiting for SSH..."

# Ensure VM is running
gcloud compute instances start "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true

# Make sure IAP firewall rule exists (idempotent)
if ! gcloud compute firewall-rules describe allow-iap-ssh --project "${PROJECT_ID}" >/dev/null 2>&1; then
  gcloud compute firewall-rules create allow-iap-ssh \
    --project="${PROJECT_ID}" --network=default --direction=INGRESS --action=ALLOW \
    --rules=tcp:22 --source-ranges=35.235.240.0/20
fi

# Ensure caller has IAP TCP privilege
ACCOUNT="$(gcloud config get-value account)"
gcloud projects add-iam-policy-binding "${PROJECT_ID}" \
  --member="user:${ACCOUNT}" --role="roles/iap.tunnelResourceAccessor" >/dev/null || true

# Try IAP SSH first (with retries)
USE_IAP=0
for i in {1..8}; do
  if gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
        --tunnel-through-iap --command "echo VM up" >/dev/null 2>&1; then
    USE_IAP=1; break
  fi
  echo "SSH (IAP) not ready yet... retry $i/8"; sleep 8
done

if [[ "${USE_IAP}" -eq 0 ]]; then
  echo "IAP failed | attaching temporary external IP and retrying classic SSH..."
  gcloud compute instances add-access-config "${INSTANCE_NAME}" \
    --zone "${ZONE}" --project "${PROJECT_ID}" >/dev/null || true
  if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
         --command "echo VM up" >/dev/null 2>&1; then
    echo "ERROR: SSH failed even after attaching external IP."
    echo "Hint: check serial console:"
    echo "  gcloud compute instances get-serial-port-output ${INSTANCE_NAME} --zone ${ZONE} --project ${PROJECT_ID} --port=1 | tail -n 200"
    exit 1
  fi
  SSH_EXTRAS=()   # public-ip path
else
  SSH_EXTRAS=(--tunnel-through-iap)
fi
```

```
# helper for later
ssh_run() {
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" \
    "${SSH_EXTRAS[@]}" --command "$1"
}



gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command "echo VM up" >/dev/null

# -------- Install runtimes (Qt5, TCL, newer libstdc++) --------
echo "Installing Qt5/TCL and modern libstdc++ ..."
gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
  "set -e; sudo apt-get update -y && sudo apt-get install -y \
   software-properties-common ca-certificates build-essential unzip \
   libstdc++6 libgcc-s1 \
   libqt5widgets5 libqt5gui5 libqt5core5a libqt5printsupport5 libqt5svg5 \
   libxkbcommon-x11-0 libxrender1 libx11-xcb1 libxext6 libsm6 libice6 \
   libxcb1 libxcb-render0 libxcb-shape0 libxcb-xfixes0 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 \
   libopengl0 libgl1 libglu1-mesa \
   tcl tcl8.6 tk tk8.6"

# Ensure GLIBCXX_3.4.32 present; upgrade if needed
if ! gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
  "strings /usr/lib/x86_64-linux-gnu/libstdc++.so.6 | grep -q 'GLIBCXX_3\.4\.32'"; then
  echo "Upgrading libstdc++ from Ubuntu Toolchain PPA..."
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
    "sudo add-apt-repository -y ppa:ubuntu-toolchain-r/test && sudo apt-get update -y && sudo apt-get install -y libstdc++6"
  # Re-check
  gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
    "strings /usr/lib/x86_64-linux-gnu/libstdc++.so.6 | grep -q 'GLIBCXX_3\.4\.32' && echo 'libstdc++ OK' || (echo 'ERROR: GLIBCXX_3.4.32 still missing'; exit
fi

# -------- Prep workdir --------
gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
  "sudo mkdir -p ${WORKDIR}/in ${WORKDIR}/out && sudo chown -R \$USER:\$USER ${WORKDIR}"

# -------- Copy ALL local files to VM --------
echo "Uploading ALL files from current directory to VM..."
gcloud compute scp --recurse --zone "${ZONE}" --project "${PROJECT_ID}" \
  ./ "${INSTANCE_NAME}:${WORKDIR}/in/"

# Ensure LVR_batch is executable if present
gcloud compute ssh "${INSTANCE_NAME}" --zone "${ZONE}" --project "${PROJECT_ID}" --command \
  "chmod +x ${WORKDIR}/in/LVR_batch || true"

# -------- Run exactly: LVR_batch jpeg_sky.cmd --------

gcloud compute ssh lvr-node-1 --zone us-central1-a --project jpeg-lvr-gcp --command \
'cd /opt/ramtera/lvr_run/in && \
 export LD_LIBRARY_PATH=/usr/lib/x86_64-linux-gnu:${LD_LIBRARY_PATH:-}; \
 export QT_QPA_PLATFORM=offscreen; \
 ./LVR_batch jpeg_sky.cmd'

echo "Done. Running"
```

12 Video Links

12.1 Video of VM setup and installation of ubuntu packages:

- **Video of VM setup and installation of ubuntu packages:** Open link

12.2 Video of running application:

- **Video of running application** Open link

12.3 Video of files copied from VM to Local:

- **Video of files copied from VM to Local:** Open link

12.4 Video of Close VM:

- **Video of Close VM:** Open link

12.5 Video of checking the VM status:

- **Video of checking the VM status:** Open link

13 Screen Shots

13.1 Design files copied to VM

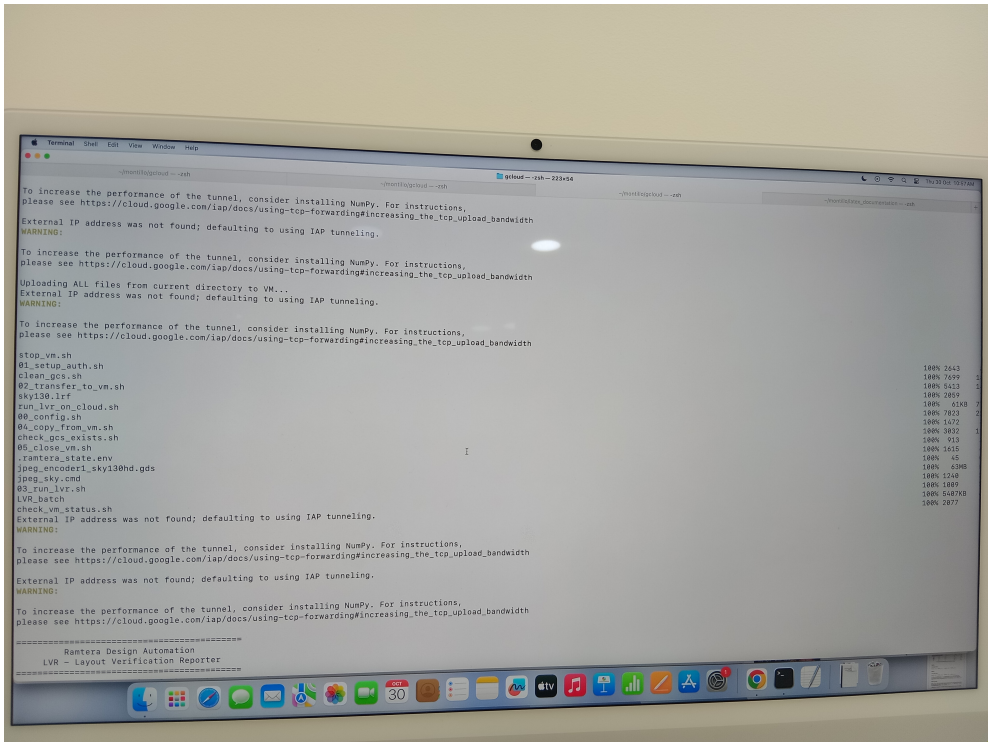

Figure 1: Design files copied to VM

13.2 Ramtera LVR application start

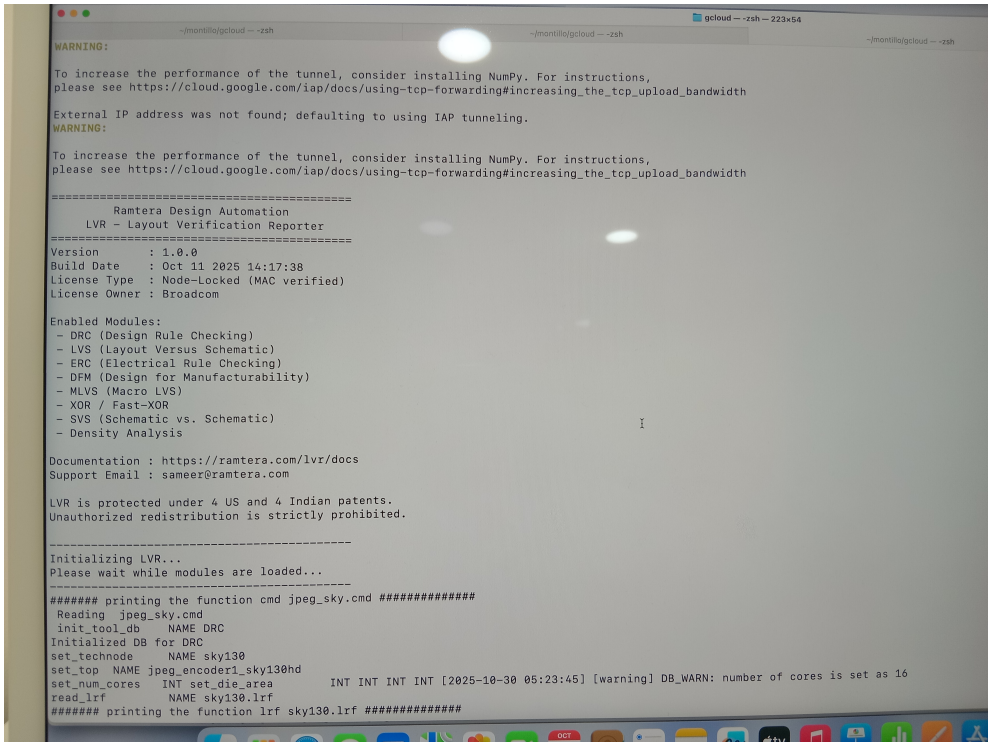

Figure 2: Ramtera LVR application start

13.3 Ramtera LVR application completed

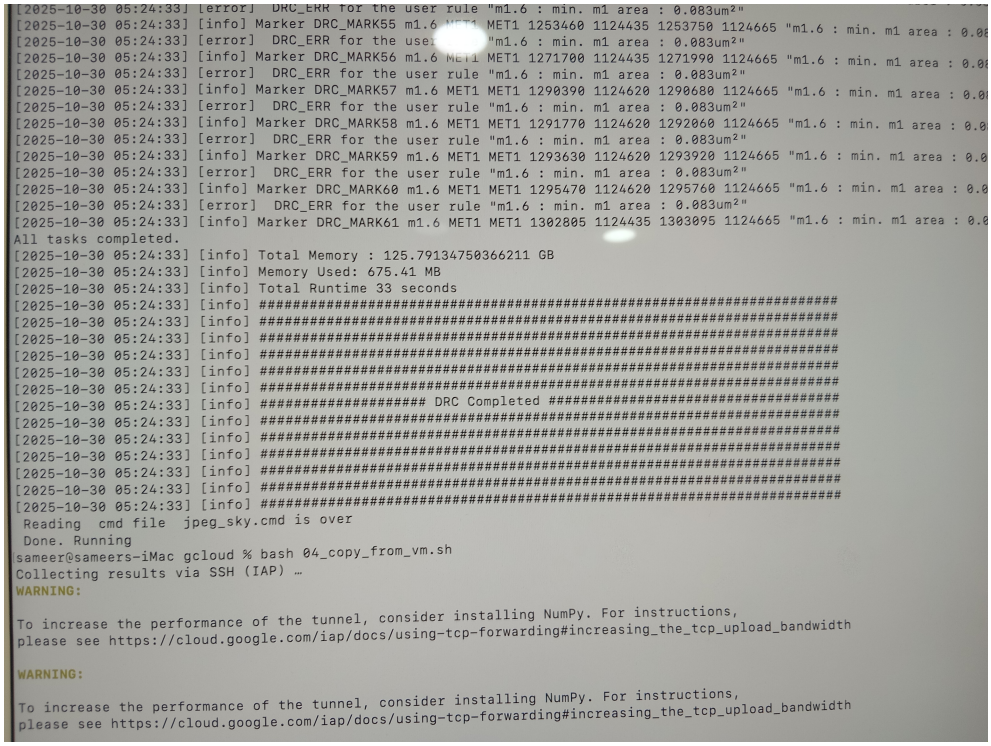

Figure 3: Ramtera LVR application completed

13.4 Stopped and terminated VM

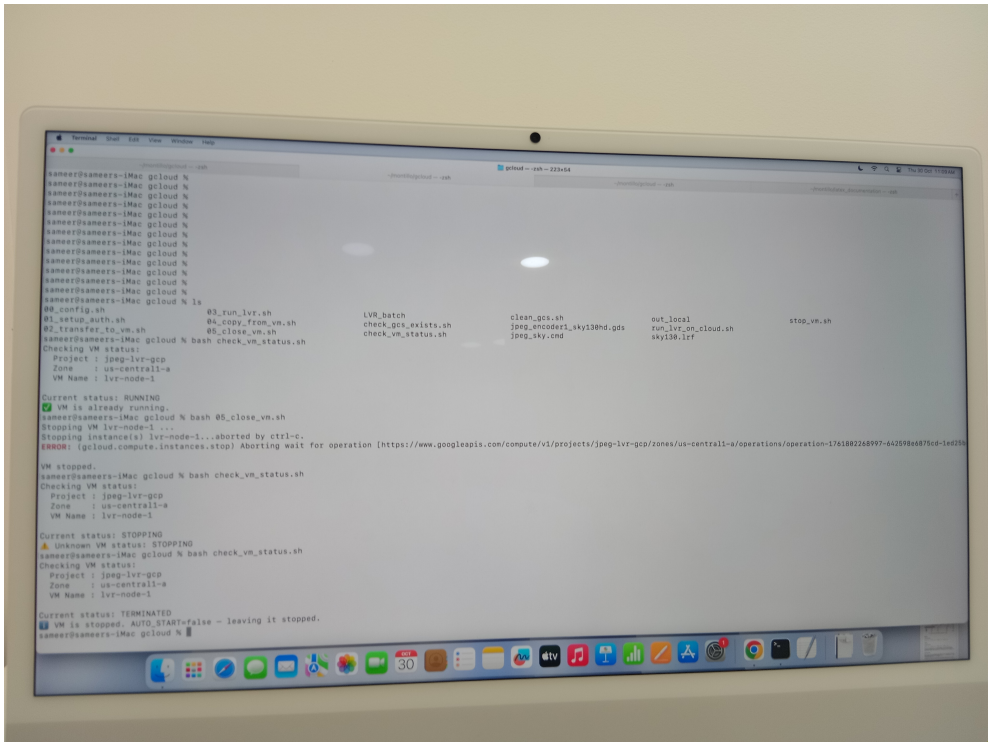

Figure 4: Stopped and terminated VM

13.5 Local results files copied from VM

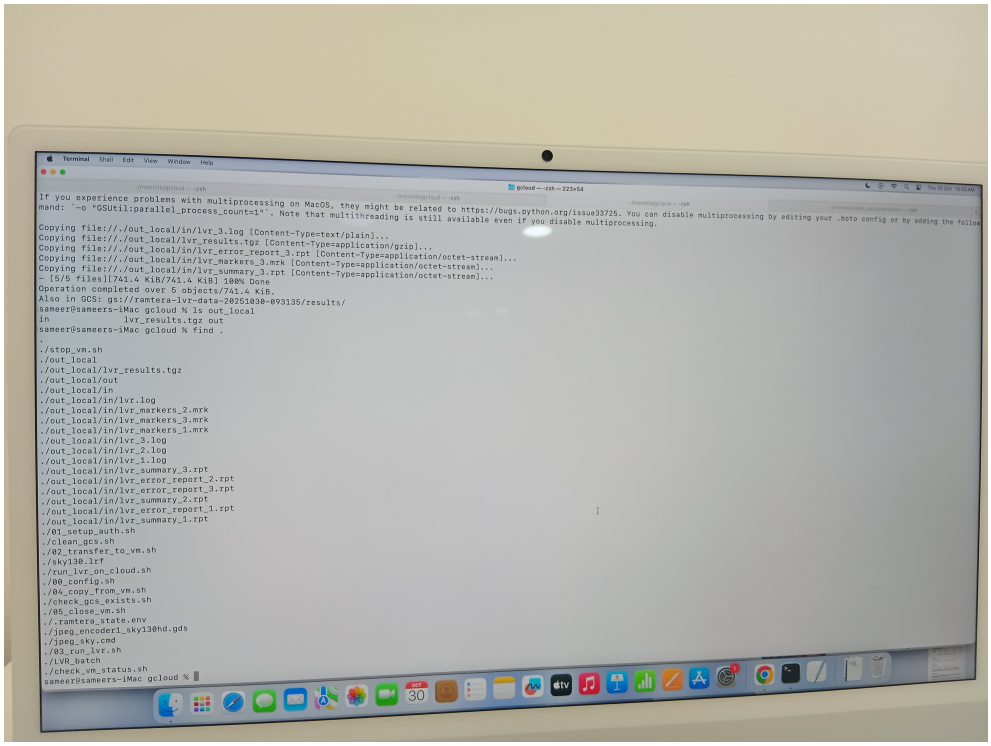

Figure 5: Local results files copied from VM

14 Benefits of Google Cloud Application

- Scalability: Right-size compute for small blocks to full-chip verification; parallel tiling across VMs.
- Reliability: Durable, globally replicated storage for design artifacts via GCS.
- Cost Control: Use Spot VMs and automated teardown to minimize cost.
- Repeatability: Scripted, auditable runs ideal for Partner validation and future Marketplace listing.