第一种情况可能遇到，可以在这台计算机上安装开发工具或运行开发环境，这是开发者最容易接受的方案。例如，在本地安装 Python 解释器、Node.js 运行时、Docker 容器，或配置 Conda 环境。

例如，可能要 Windows 或 Mac 用户安装 Python 3 或 Node.js。

这种选择通常合理，因为这是工业开发方案，开发者通常会为此付出成本。

但是，这对普通用户来说是不可行的。用户通常会期望下载一个“Setup”安装程序或应用。

输入端口

第二个问题，是一个用户工作流程。如果不安装任何工具，用户通常会期望在浏览器中打开一个网址，或双击一个应用程序。

输出端口

即使解决了运行时问题，仍需要为应用程序提供一个输出端口。

例如，一个网址。

结论

核心	一个
解决方	分别解
问题	运行时
解决方	为了解决运行时问题，可以采用云端开发环境（如 Codespaces）。这种方案通常会要求用户注册一个新的开发平台，并学习如何使用该平台的工作流程。这对于一个只想运行一个简单工具的用户来说，成本过高。
核心	一个
解决方	为了解决输出端问题，可以将应用程序打包成一个可执行文件（.exe 或 .app）。这样，用户可以双击文件来启动应用，而无需打开浏览器。
问题	输出端口的核心问题是如何将一个本地应用程序的功能，以一种简单、标准化的方式，提供给最终用户。这涉及到将应用程序打包成一个可执行文件，并确保其能在目标操作系统上顺利运行。
解决方	为了解决输出端问题，可以使用 Electron 或 Tauri 等框架，将 Web 应用程序打包成一个桌面应用。这样，用户可以双击一个图标来启动应用，而无需打开浏览器。

最佳解决方案

最佳解决方案是将 Web 应用程序打包成一个桌面应用。这种方案通过将应用程序的运行时环境与应用本身一起打包，从而解决了运行时问题。同时，它提供了一个标准化的、用户友好的启动方式，从而解决了输出端问题。

结论

这种方案在技术上可行，并能为用户提供良好的体验。

2025年7月1日

我上个月做了 Mac、Web、Linux、

工具和应用程序上线不到两个月就成功实现了

最新版本工具中，最新版本的第一周，我们收到了超过 [illegible] 次的下载量。这一数字对于我们这样的个人项目来说，是一个巨大的成就。我们还为 Windows 版本开发了工具。

这次，我们在这个版本中做了两个改动。这一改变是一个重大的决策，也解决了一个问题。这会影响最大用户群体的两个方面。首先，是安装。

问题安装程序

```
# Python 3.9 or newer required.
import os, sys, shutil, urllib.request,
import zipfile, tarfile, subprocess,
import platform, ssl, time, hashlib,
import pathlib, textwrap, traceback,

# ---------------------------------------
# 配置区域
# ---------------------------------------
PYTHON_EMBED_URL = "https://www.python.org/ftp/
PYTHON_EMBED_SHA256 = "a1b2c3d4e5f6..."  # 可选
PYTHON_EMBED_ZIP = "python_embed.zip"
VENV_DIR = "venv"
REQUIREMENTS = "requirements.txt"
MAIN_CMD = "-m myapp.main"
LOG_FILE = "bootstrap.log"

# ---------------------------------------
# 日志工具
# ---------------------------------------
def log(msg):
    ts = time.strftime("%H:%M:%S")
    line = f"[{ts}] {msg}"
    print(line, flush=True)
    try:
        with open(LOG_FILE, "a", encoding="utf-8"
            f.write(line + "\n")
    except Exception:
        pass

def fatal(msg):
    log("FATAL: " + msg)
    sys.exit(1)

# ---------------------------------------
# 平台检测
# ---------------------------------------
def is_windows():
    return os.name == "nt"

def is_mac():
    return sys.platform == "darwin"

def is_linux():
    return sys.platform.startswith("linux")

def platform_tag():
    return f"{platform.system()}-{platform.mach
    # 例如 "Windows-AMD64" 或 "Linux-x86_64"

# ---------------------------------------
# 下载工具
# ---------------------------------------
def download(url, dest):
    log("下载 " + url)
    ctx = ssl.create_default_context()
    try:
        with urllib.request.urlopen(url, timeo
            total = int(resp.headers.get("Cont
            chunk = 8192
            while True:
                data = resp.read(chunk)
                if not data:
                    break
                f.write(data)
                done += len(data)
                if total > 0:
                    pct = done * 100 // total
                    sys.stdout.write("\r  %d%%
                    sys.stdout.flush()
            sys.stdout.write("\n")
    except urllib.error.URLError as e:
        fatal("下载失败: " + str(e))

def verify_sha256(path, expected):
    if not expected:
        return True
    h = hashlib.sha256()
    with open(path, "rb") as f:
        while True:
            b = f.read(65536)
            if not b:
                break
            h.update(b)
    ok = h.hexdigest().lower() == expected.lower
    log("SHA256 " + ("OK" if ok else "MISMATCH"
    return ok

# ---------------------------------------
# 解压工具
# ---------------------------------------
def extract_zip(zipfile, dest):
    log("解压 " + zipfile + " → " + dest)
    with zipfile.ZipFile(zipfile, "r") as z:
        for m in z.namelist():
            z.extract(m, dest)

def extract_tar(tarfile, dest):
    log("解压 " + tarfile + " → " + dest)
    with tarfile.open(tarfile, "r:*") as t:
        try:
            t.extractall(dest, filter="data"
            pass
        except TypeError:
            t.extractall(dest)

# ---------------------------------------
# 目录工具
# ---------------------------------------
def ensure_dir(p):
    os.makedirs(p, exist_ok=True)

def rmtree(p):
    if os.path.exists(p):
        log("删除 " + p)
        shutil.rmtree(p, ignore_errors=True)

# ---------------------------------------
# 嵌入式 Python 安装
# ---------------------------------------
def install_embed_python(base, zipfile, expect
    log("安装嵌入式 Python → " + base)
    ensure_dir(base)
    if not os.path.exists(zipfile):
        fatal("找不到 " + zipfile + "，请先下载")
    if not verify_sha256(zipfile, expect):
        log("警告：SHA256 不匹配，继续尝试")
    extract_zip(zipfile, base)
    return os.path.join(base, "python.exe")

# ---------------------------------------
# 修补 ._pth 文件
# ---------------------------------------
def patch_pth(base):
    pth = None
    for name in os.listdir(base):
        if name.endswith("._pth"):
            pth = os.path.join(base, name)
            break
    if not pth:
        log("未找到 ._pth 文件，跳过")
        return False
    log("修补 " + pth)
    with open(pth, "r", encoding="utf-8") as f
        lines = f.read().splitlines()
    out = []
    for ln in lines:
        if ln.strip() == "#import site":
            out.append("import site")
            log("  启用 import site")
        else:
            out.append(ln)
    with open(pth, "w", encoding="utf-8") as f
        f.write("\n".join(out) + "\n")
    return True

# ---------------------------------------
# 安装 pip
# ---------------------------------------
def install_pip(base, py):
    log("安装 pip")
    get = os.path.join(base, "get-pip.py"
    if not os.path.exists(get):
        download("https://bootstrap.pypa.io/
    rc = run([py, get, "--no-warn-script-loc
    if rc != 0:
        fatal("pip 安装失败")
    return rc

# ---------------------------------------
# 子进程工具
# ---------------------------------------
def run(cmd, cwd=None, env=None):
    log("$ " + " ".join(str(c) for c in cmd)
    try:
        p = subprocess.Popen(cmd, cwd=cwd, e
        for line in iter(p.stdout.readline,
            sys.stdout.write("  | " + line.
            sys.stdout.flush()
        p.stdout.close()
        return p.wait()
    except OSError as e:
        log("启动失败: " + str(e))
        return 127

def run_capture(cmd, cwd=None, env=None):
    try:
        p = subprocess.Popen(cmd, cwd=cwd, e
        out, err = p.communicate()
        return p.returncode, (out or b"").de
                             (err or b"").de
    except OSError:
        return 127, "", ""

# ---------------------------------------
# 虚拟环境
# ---------------------------------------
def venv_python(base):
    if is_windows():
        return os.path.join(base, "Scripts"
    return os.path.join(base, "bin", "pytho

def create_venv(base, py):
    if os.path.exists(venv_python(base)):
        log("虚拟环境已存在，跳过创建")
        return 0
    log("创建虚拟环境 → " + base)
    rc = run([py, "-m", "venv", base])
    if rc != 0:
        log("venv 失败，尝试 ensurepip")
        ensure_dir(base)
        rc = run([py, "-m", "ensurepip", "
                  "--root", base])
    return rc

def pip_install(base, req):
    if not os.path.exists(req):
        log("没有 " + req + "，跳过")
        return 0
    py = venv_python(base)
    log("pip 安装 -r " + req)
    rc = run([py, "-m", "pip", "install"
              "--disable-pip-version-check"
              "--no-warn-script-location"
              "-r", req])
    if rc != 0:
        log("pip 失败 rc=" + str(rc))
    return rc

# ---------------------------------------
# 主流程
# ---------------------------------------
def main():
    log("=" * 40)
    log("启动 " + time.strftime("%Y-%m-%d %H
    log("平台: " + platform_tag())
    log("Python: " + sys.version.replace("\
    log("工作目录: " + os.getcwd())

    base = os.path.abspath(VENV_DIR)
    zipfile = os.path.abspath(PYTHON_EMBED_
    req = os.path.abspath(REQUIREMENTS)

    py = install_embed_python(base, zipfile
                              PYTHON_EMBED_
    if not os.path.exists(py):
        fatal("嵌入式 Python 安装失败")

    patch_pth(base)

    rc = install_pip(base, py)

    if os.path.exists(req):
        rc = pip_install(base, req)

    vpy = venv_python(base)
    if os.path.exists(vpy):
        rc = create_venv(base, py)
        if rc == 0:
            py = vpy

    log("启动 " + MAIN_CMD)
    rc = run([py, *MAIN_CMD.split()])
    log("应用退出 rc=" + str(rc))
    sys.exit(rc)

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        log("被中断")
        sys.exit(130)
    except Exception:
        log("未捕获异常")
        traceback.print_exc()
        sys.exit(2)
```