
Perspt Documentation

Release 0.4.4

Ronak Rathore, Vikrant Rathore


Dec 23, 2025

□ Getting Started

1	Key Features	3
1.1	Introduction to Perspt	3
1.2	Quick Start	5
1.3	Installation Guide	7
1.4	Getting Started	17
1.5	Tutorials	28
1.6	User Guide	39
1.7	Concepts	80
1.8	How-To Guides	87
1.9	Configuration Guide	97
1.10	Reference	109
1.11	API Reference	117
1.12	Developer Guide	140
1.13	Changelog	152
1.14	License	159
1.15	Acknowledgments	163
2	Quick Links	167
3	Indices	169
	Index	171

Your Terminal's Window to the AI World

Perspt is a high-performance terminal-based LLM interface with autonomous coding capabilities powered by the **SRBN (Stabilized Recursive Barrier Network)** engine.

 **Quick Start** Install and chat in 5 minutes.




Quick Start  **Agent Mode** Autonomous code generation with SRBN.

Agent Mode Tutorial  **Architecture** 6-crate workspace design.

Architecture

Chapter 1

Key Features

	SRBN Agent Mode — Autonomous coding with Lyapunov stability guarantees (v0.5.0)
<input type="checkbox"/>	Multi-Provider — OpenAI GPT-5.2, Claude Opus 4.5, Gemini 3, Groq, Ollama
<input type="checkbox"/>	LSP Integration — Real-time type checking via <code>ty</code> server
<input type="checkbox"/>	Test Runner — pytest integration with <code>V_log</code> energy
<input type="checkbox"/>	Token Budget — Cost control with usage monitoring
	Beautiful TUI — Ratatui-based with diff viewer and task tree
	Security — Policy engine with command sanitization

1.1 Introduction to Perspt

1.1.1 What is Perspt?

Perspt (pronounced “perspect,” short for **P**ersonal **S**pectrum **P**ertaining **T**houghts) is a high-performance, terminal-based interface to Large Language Models with **autonomous coding capabilities**.




Version 0.5.0 Highlights

- **SRBN Agent Mode** — Autonomous coding with Lyapunov stability guarantees
- **6-Crate Architecture** — Modular, extensible workspace design
- **LSP Integration** — Real-time type checking with `ty` server
- **Latest Models** — GPT-5.2, Claude Opus 4.5, Gemini 3

1.1.2 Architecture

Perspt is built as a **6-crate Rust workspace**:

1.1.3 Key Features

	SRBN Agent Mode	Autonomous coding with stability guarantees. Decomposes tasks, generates code, verifies with LSP.
<input type="checkbox"/>	Multi-Provider	OpenAI GPT-5.2, Anthropic Claude Opus 4.5, Google Gemini 3, Groq, Cohere, XAI, DeepSeek, Ollama.
<input type="checkbox"/>	LSP Integration	Real-time Python type checking using ty server. Computes syntax energy V_{syn} .
<input type="checkbox"/>	Test Runner	pytest integration with weighted failure scoring for logic energy V_{log} .
<input type="checkbox"/>	Merkle Ledger	Git-style change tracking with rollback support.
	Security	Starlark policy rules and command sanitization.
<input type="checkbox"/>	Token Budget	Cost tracking with per-request limits.
	Beautiful TUI	Ratatui-based with markdown rendering, diff viewer, task tree.

1.1.4 SRBN: Stabilized Recursive Barrier Network

The core innovation in Perspt v0.5.0 is the SRBN control loop:

Lyapunov Energy: $V(x) = \alpha \cdot V_{\text{syn}} + \beta \cdot V_{\text{str}} + \gamma \cdot V_{\text{log}}$

- **V_{syn} :** LSP diagnostics (errors, warnings)
- **V_{str} :** Structural analysis
- **V_{log} :** Test failures (weighted by criticality)

1.1.5 CLI Commands

Command	Description	Example
chat	Interactive TUI	<code>perspt chat</code>
agent	Autonomous coding	<code>perspt agent "create calculator"</code>
init	Project setup	<code>perspt init --memory</code>
config	Configuration	<code>perspt config --show</code>
ledger	Change history	<code>perspt ledger --recent</code>
status	Agent status	<code>perspt status</code>
abort	Cancel session	<code>perspt abort</code>
resume	Resume session	<code>perspt resume</code>

1.1.6 Supported Providers

Provider	Environment Variable	Models (2025)
OpenAI	OPENAI_API_KEY	GPT-5.2, o3-mini, o1-preview
Anthropic	ANTHROPIC_API_KEY	Claude Opus 4.5
Google	GEMINI_API_KEY	Gemini 3 Flash, Gemini 3 Pro
Groq	GROQ_API_KEY	Llama 3.x (ultra-fast)
Cohere	COHERE_API_KEY	Command R+
XAI	XAI_API_KEY	Grok
DeepSeek	DEEPSEEK_API_KEY	DeepSeek Coder
Ollama	(none)	Local models

1.1.7 Philosophy


*The keyboard hums, the screen aglow,
AI's wisdom, a steady flow.
Through SRBN's loop, stability we find,
Code that works, refined and aligned.*

—The Perspt Manifesto

Perspt embodies the belief that AI tools should be:

- **Fast** — Rust-native performance
- **Stable** — Lyapunov-guaranteed convergence
- **Secure** — Policy-controlled execution
- **Extensible** — Modular crate architecture

1.1.8 Next Steps

 **Quick Start** Get running in 5 minutes.

Quick Start  **Agent Mode** Autonomous code generation.

Agent Mode Tutorial  **Architecture** Understand the 6-crate design.

Architecture

1.2 Quick Start

Get Perspt running in 5 minutes.

1.2.1 Prerequisites

Rust 1.82+	rustup.rs for building from source
API Key	From any provider (OpenAI, Anthropic, Google, etc.) OR Ollama for local models

1.2.2 Installation

From Source (Recommended)

```
git clone https://github.com/eonseed/perspt.git
cd perspt
cargo build --release
```

Cargo Install

```
cargo install perspt
```

With Ollama (No API Key)

```
# Install Ollama
brew install ollama # macOS

# Start and pull a model
```

(continues on next page)

(continued from previous page)

```
ollama serve
ollama pull llama3.2
```

1.2.3 Set Your API Key

```
# Choose your provider
export OPENAI_API_KEY="sk-..." # OpenAI
export ANTHROPIC_API_KEY="sk-ant-..." # Anthropic
export GEMINI_API_KEY="..." # Google
```

1.2.4 Run Your First Chat

```
# Start the TUI
./target/release/perspt

# Or with a specific model
perspt chat --model gpt-5.2
```

Type your message and press **Enter**. Press **Esc** to exit.

1.2.5 Try Agent Mode

Added in version 0.5.0.

Let Perspt autonomously write code:

```
# Basic task
perspt agent "Create a Python calculator with add, subtract, multiply, divide"

# With workspace directory
perspt agent -w ./my-project "Add unit tests for the API"

# Auto-approve all changes
perspt agent -y "Refactor the parser for better error handling"
```

The SRBN engine will:

1. **Sheafify** — Decompose task into sub-tasks
2. **Speculate** — Generate code for each sub-task
3. **Verify** — Check with LSP and tests
4. **Converge** — Retry until $V(x) < \epsilon$
5. **Commit** — Record in Merkle ledger

See also

Agent Mode Tutorial for a full walkthrough.

1.2.6 Choose Your Mode

Mode	Command	Best For
Chat TUI	<code>perspt</code> or <code>perspt chat</code>	Interactive conversations with markdown rendering
Agent	<code>perspt agent "<task>"</code>	Autonomous code generation and modification
Status	<code>perspt status</code>	Check current agent session status

1.2.7 Essential Commands

```
# Configuration
perspt config --show          # View current config
perspt config --edit          # Edit in $EDITOR
perspt init --memory --rules  # Initialize project


# Agent management
perspt status                  # Current session status
perspt abort                   # Cancel current session
perspt resume                  # Resume interrupted session


# Change tracking
perspt ledger --recent         # View recent changes
perspt ledger --rollback abc   # Rollback to commit
```

1.2.8 Key Bindings (Chat TUI)


Key	Action
Enter	Send message
Esc	Exit application
↑/↓	Scroll chat history
Page Up/Down	Fast scroll
/save	Save conversation (command)

1.2.9 Next Steps

 **Tutorials** Step-by-step learning guides.

Tutorials  **Configuration** Customize providers and models.

Configuration  **Agent Deep Dive** Master autonomous coding.

Agent Mode Tutorial  **Architecture** Understand the 6-crate design.

Architecture

1.3 Installation Guide

This comprehensive guide covers all the ways to install Perspt on your system, from simple binary downloads to building from source.

1.3.1 System Requirements

Minimum Requirements


Component	Requirement
Operating System	Linux, macOS, Windows 10+
Architecture	x86_64, ARM64 (Apple Silicon)
Memory	50MB RAM minimum
Storage	10MB disk space
Terminal	Any terminal with UTF-8 support
Network	Internet connection for AI API calls

Recommended Requirements


Component	Recommendation
Terminal	Modern terminal with 256+ colors and Unicode support
Font	Monospace font with good Unicode coverage (e.g., Fira Code, JetBrains Mono)
Shell	Bash, Zsh, Fish, or PowerShell
Memory	100MB+ RAM for optimal performance

1.3.2 Quick Install

Choose your preferred installation method:

 Binary Download (Fastest)

 Cargo Install

 ☐ Build from Source

Download pre-built binaries for immediate use:

Linux x86_64:

```
curl -L https://github.com/eonseed/perspt/releases/latest/download/perspt-linux-x86_64.tar.  
gz | tar xz  
chmod +x perspt  
sudo mv perspt /usr/local/bin/
```

macOS (Intel):

```
curl -L https://github.com/eonseed/perspt/releases/latest/download/perspt-darwin-x86_64.tar.  
gz | tar xz  
chmod +x perspt  
sudo mv perspt /usr/local/bin/
```

macOS (Apple Silicon):

```
curl -L https://github.com/eonseed/perspt/releases/latest/download/perspt-darwin-arm64.tar.  
gz | tar xz  
chmod +x perspt  
sudo mv perspt /usr/local/bin/
```

Windows:

```
# Download from GitHub releases page
# Extract perspt.exe and add to PATH
```

Install using Rust's package manager:

```
# Install from crates.io
cargo install perspt

# Or install the latest development version
cargo install --git https://github.com/eonseed/perspt
```

Build the latest version from source:

```
# Clone repository
git clone https://github.com/eonseed/perspt.git
cd perspt

# Build release version
cargo build --release

# Install to cargo bin
cargo install --path .
```

1.3.3 Package Managers

Homebrew (macOS/Linux)

```
# Add tap (when available)
brew tap eonseed/perspt

# Install
brew install perspt

# Update
brew upgrade perspt
```

Scoop (Windows)

```
# Add bucket (when available)
scoop bucket add perspt https://github.com/eonseed/scoop-perspt

# Install
scoop install perspt

# Update
scoop update perspt
```

Chocolatey (Windows)

```
# Install (when available)
choco install perspt
```

(continues on next page)

(continued from previous page)

```
# Update
choco upgrade perspt
```

APT (Debian/Ubuntu)

```
# Add repository (when available)
curl -fsSL https://releases.perspt.dev/gpg | sudo gpg --dearmor -o /usr/share/keyrings/
↳perspt.gpg
echo "deb [signed-by=/usr/share/keyrings/perspt.gpg] https://releases.perspt.dev/apt
↳stable main" | sudo tee /etc/apt/sources.list.d/perspt.list

# Install
sudo apt update
sudo apt install perspt
```

RPM (Red Hat/Fedora)

```
# Add repository (when available)
sudo dnf config-manager --add-repo https://releases.perspt.dev/rpm/perspt.repo

# Install
sudo dnf install perspt
```

1.3.4 Building from Source

Prerequisites

```
# Install Rust (if not already installed)
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env

# Verify installation
rustc --version
cargo --version
```

Clone and Build

```
# Clone the repository
git clone https://github.com/eonseed/perspt.git
cd perspt

# Build in release mode
cargo build --release

# The binary will be in target/release/perspt
./target/release/perspt --version
```

Install System-Wide

```
# Option 1: Using cargo install
cargo install --path .

# Option 2: Manual installation
sudo cp target/release/perspt /usr/local/bin/
sudo chmod +x /usr/local/bin/perspt

# Option 3: User-local installation
mkdir -p ~/.local/bin
cp target/release/perspt ~/.local/bin/
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Development Build

For development and testing:

```
# Clone with all development tools
git clone https://github.com/eonseed/perspt.git
cd perspt

# Install development dependencies
cargo install cargo-watch cargo-edit

# Build in debug mode
cargo build

# Run tests
cargo test

# Run with hot reload during development
cargo watch -x run
```

1.3.5 Docker Installation

Run Perspt in a Docker container:

Basic Usage

```
# Pull the image
docker pull ghcr.io/eonseed/perspt:latest

# Run interactively
docker run -it --rm \
  -e OPENAI_API_KEY="$OPENAI_API_KEY" \
  ghcr.io/eonseed/perspt:latest
```

With Configuration

```
# Create a config directory
mkdir -p ~/.config/perspt

# Create your config.json
cat > ~/.config/perspt/config.json << EOF
{
  "api_key": "your-api-key-here",
  "default_model": "gpt-4o-mini",
  "default_provider": "openai"
}
EOF

# Run with mounted config
docker run -it --rm \
  -v ~/.config/perspt:/app/config \
  ghcr.io/eonseed/perspt:latest \
  --config /app/config/config.json
```

Docker Compose

Create a *docker-compose.yml* file:

```
version: '3.8'
services:
  perspt:
    image: ghcr.io/eonseed/perspt:latest
    stdin_open: true
    tty: true
    environment:
      - OPENAI_API_KEY=${OPENAI_API_KEY}
    volumes:
      - ./config:/app/config
    command: ["--config", "/app/config/config.json"]
```

Run with:

```
docker-compose run --rm perspt
```

1.3.6 Platform-Specific Instructions

Linux

Ubuntu/Debian:

```
# Update package list
sudo apt update

# Install dependencies for building (if building from source)
sudo apt install build-essential curl git

# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

(continues on next page)

(continued from previous page)

```
source $HOME/.cargo/env
```

```
# Install Perspt
cargo install perspt
```

Arch Linux:

```
# Install from AUR (when available)
yay -S perspt

# Or build from source
sudo pacman -S rust git
git clone https://github.com/eonseed/perspt.git
cd perspt
cargo build --release
```

CentOS/RHEL/Fedora:

```
# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env

# Install development tools
sudo dnf groupinstall "Development Tools"
sudo dnf install git

# Install Perspt
cargo install perspt
```

macOS**Using Homebrew (Recommended):**

```
# Install Homebrew if not already installed
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install Rust
brew install rust

# Install Perspt
cargo install perspt
```

Using MacPorts:

```
# Install Rust
sudo port install rust

# Install Perspt
cargo install perspt
```

Manual Installation:

```
# Install Xcode command line tools
xcode-select --install

# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source $HOME/.cargo/env

# Install Perspt
cargo install perspt
```

Windows

Using Chocolatey:

```
# Install Chocolatey
Set-ExecutionPolicy Bypass -Scope Process -Force
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.
↪ServicePointManager]::SecurityProtocol -bor 3072
iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/
↪install.ps1'))

# Install Rust
choco install rust

# Install Perspt
cargo install perspt
```

Using Scoop:

```
# Install Scoop
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
irm get.scoop.sh | iex

# Install Rust
scoop install rust

# Install Perspt
cargo install perspt
```

Manual Installation:

1. Download and install Rust from rustup.rs
2. Open Command Prompt or PowerShell
3. Run: `cargo install perspt`

1.3.7 Verification

After installation, verify that Perspt is working correctly:

```
# Check version
perspt --version

# Check help
perspt --help
```

(continues on next page)

(continued from previous page)

```
# Test basic functionality (requires API key)
export OPENAI_API_KEY="your-key-here"
perspt --model-name gpt-4o-mini
```

You should see output similar to:

```
perspt 0.4.0
Your Terminal's Window to the AI World
```

1.3.8 Updating Perspt

Cargo Installation

```
# Update to latest version
cargo install perspt --force

# Or update all cargo packages
cargo install-update -a
```

Binary Installation

```
# Download and replace binary
curl -L https://github.com/eonseed/perspt/releases/latest/download/perspt-linux-x86_64.tar.
gz | tar xz
sudo mv perspt /usr/local/bin/
```

Package Managers

```
# Homebrew
brew upgrade perspt

# APT
sudo apt update && sudo apt upgrade perspt

# DNF
sudo dnf upgrade perspt

# Chocolatey
choco upgrade perspt

# Scoop
scoop update perspt
```

1.3.9 Uninstallation

Cargo Installation

```
# Uninstall using cargo
cargo uninstall perspt
```

Manual Binary

```
# Remove binary
sudo rm /usr/local/bin/perspt

# Remove configuration (optional)
rm -rf ~/.config/perspt
```

Package Managers

```
# Homebrew
brew uninstall perspt

# APT
sudo apt remove perspt

# DNF
sudo dnf remove perspt

# Chocolatey
choco uninstall perspt

# Scoop
scoop uninstall perspt
```

1.3.10 Troubleshooting

Common Issues

“Command not found” error:

```
# Check if cargo bin is in PATH
echo $PATH | grep -q "$HOME/.cargo/bin" && echo "Cargo bin in PATH" || echo "Cargo bin NOT
↳ in PATH"

# Add to PATH if missing
echo 'export PATH="$HOME/.cargo/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

Permission denied:

```
# Make sure the binary is executable
chmod +x /usr/local/bin/perspt

# Or use without sudo
mkdir -p ~/.local/bin
cp perspt ~/.local/bin/
export PATH="$HOME/.local/bin:$PATH"
```

Build failures:

```
# Update Rust toolchain
rustup update
```

(continues on next page)

(continued from previous page)

```
# Clear cargo cache
cargo clean

# Rebuild
cargo build --release
```

Missing dependencies on Linux:

```
# Ubuntu/Debian
sudo apt install build-essential pkg-config libssl-dev

# CentOS/RHEL/Fedora
sudo dnf groupinstall "Development Tools"
sudo dnf install pkgconfig openssl-devel
```

Getting Help

If you encounter issues during installation:

1. **Check the GitHub Issues:** [Issues Page](#)
2. **Join the Discussion:** [GitHub Discussions](#)
3. **Read the FAQ:** [Troubleshooting](#)
4. **Contact Support:** Create a new issue with: - Your operating system and version - Rust version (`rustc --version`) - Installation method used - Complete error message

1.3.11 Next Steps

After successful installation:

1. **Set up API keys:** [Configuration Guide](#)
2. **Learn basic usage:** [Getting Started](#)
3. **Explore features:** [User Guide](#)
4. **Join the community:** [GitHub Discussions](#)

➡ See also

- [Getting Started](#) - Your first conversation
- [Configuration Guide](#) - Setting up API keys and preferences
- [Basic Usage](#) - Everyday usage patterns
- [Troubleshooting](#) - Common issues and solutions

1.4 Getting Started

Welcome to Perspt! This guide will get you up and running with your first AI conversation in just a few minutes.

1.4.1 Prerequisites

Before installing Perspt, ensure you have the following:

System Requirements

Component	Requirement
Operating System	Linux, macOS, or Windows
Rust Toolchain	Rust 1.82.0 or later
Terminal	Any modern terminal emulator
Internet Connection	Required for AI provider API calls

API Keys

You'll need an API key from at least one AI provider:

OpenAI

Anthropic

Google

Ollama (Local)

1. Visit [OpenAI Platform](#)
2. Sign up or log in to your account
3. Navigate to API Keys section
4. Create a new API key
5. Copy and save it securely

```
export OPENAI_API_KEY="sk-your-openai-api-key-here"
```

1. Visit [Anthropic Console](#)
2. Sign up or log in
3. Go to API Keys
4. Generate a new key
5. Save it securely

```
export ANTHROPIC_API_KEY="your-anthropic-api-key-here"
```

1. Visit [Google AI Studio](#)
2. Create or select a project
3. Generate API key
4. Configure authentication

```
export GOOGLE_API_KEY="your-google-api-key-here"
```

1. Install Ollama from [ollama.ai](#)
2. Pull a model
3. Start Ollama service

```
ollama pull llama3.2
# Ollama service starts automatically
```

1.4.2 Quick Installation

Method 1: From Source (Recommended)

```
# Clone the repository
git clone https://github.com/eonseed/perspt.git
cd perspt

# Build the project
cargo build --release

# Install to your PATH (optional)
cargo install --path .

# Or run directly
./target/release/perspt
```

Method 2: Using Cargo

```
# Install from crates.io (when published)
cargo install perspt

# Run Perspt
perspt
```

Method 3: Download Binary

```
# Download the latest release (replace with actual URL)
curl -L https://github.com/eonseed/perspt/releases/latest/download/perspt-linux-x86_64.tar.gz | tar xz

# Make executable and move to PATH
chmod +x perspt
sudo mv perspt /usr/local/bin/
```

1.4.3 Your First Conversation

Let's start your first AI conversation with Perspt! You can choose between two interface modes:

1. **Interactive TUI Mode** - Rich terminal interface with markdown rendering (default)
2. **Simple CLI Mode** - Minimal command-line interface for scripting and accessibility (NEW!)

Zero-Config Quick Start

NEW! Perspt now features intelligent automatic provider detection. Simply set an environment variable for any supported provider, and Perspt will automatically detect and use it - no additional configuration needed!

Note

Automatic Provider Detection Priority:

1. OpenAI (OPENAI_API_KEY)
2. Anthropic (ANTHROPIC_API_KEY)

3. Google Gemini (GEMINI_API_KEY)
4. Groq (GROQ_API_KEY)
5. Cohere (COHERE_API_KEY)
6. XAI (XAI_API_KEY)
7. DeepSeek (DEEPSEEK_API_KEY)
8. Ollama (no API key needed - auto-detected if running)

Interactive TUI Mode (Default)

Simple CLI Mode (NEW!)

Anthropic Claude

Google Gemini

Ollama (Local)

Rich terminal interface with markdown rendering and scrollable history:

```
# Set your API key
export OPENAI_API_KEY="sk-your-actual-api-key-here"

# Launch Perspt in TUI mode (default)
perspt
# Automatically uses OpenAI with gpt-4o-mini
```

Minimal command-line interface perfect for scripting and accessibility:

```
# Set your API key
export OPENAI_API_KEY="sk-your-actual-api-key-here"

# Launch Perspt in simple CLI mode
perspt --simple-cli
# Unix-style prompt with streaming responses
```

```
# Set your API key
export ANTHROPIC_API_KEY="sk-ant-your-key"

# TUI mode (default)
perspt

# Simple CLI mode
perspt --simple-cli
# Automatically uses Anthropic with claude-3-5-sonnet-20241022
```

```
# Set your API key
export GEMINI_API_KEY="your-gemini-key"

# TUI mode (default)
perspt

# Simple CLI mode with logging
perspt --simple-cli --log-file gemini-session.txt
# Automatically uses Gemini with gemini-1.5-flash
```



```
# Just make sure Ollama is running
ollama serve

# TUI mode (default)
perspt

# Simple CLI mode for scripting
perspt --simple-cli
# Auto-detects Ollama if no other providers found
```

Step 1: Set Your API Key (Manual Configuration)

If you prefer manual configuration or want to override automatic detection:

```
# For OpenAI (most common)
export OPENAI_API_KEY="sk-your-actual-api-key-here"

# Verify it's set
echo $OPENAI_API_KEY
```

Step 2: Launch Perspt

Choose between TUI mode (rich interface) or Simple CLI mode (minimal interface):

TUI Mode (Default)

```
# Start with automatic detection (recommended)
perspt

# Or specify provider manually
perspt --provider openai --model gpt-4o-mini
```

You should see a welcome screen like this:

```

      Welcome to Perspt!
    Your Terminal's Window to AI

Provider: OpenAI
Model: gpt-4o-mini
Status: Ready

Type your message and press Enter to start chatting!
Press Ctrl+C to exit

You:
```

Simple CLI Mode (NEW!)

```
# Start simple CLI mode
perspt --simple-cli

# With session logging
perspt --simple-cli --log-file my-session.txt
```

You should see a minimal interface like this:

```
Perspt Simple CLI Mode
Model: gpt-4o-mini
Type 'exit' or press Ctrl+D to quit.

>
```

The simple CLI mode is perfect for:

- **Scripting and automation**
- **Accessibility (screen readers)**
- **Unix-style workflows**
- **Session logging**
- **Lightweight usage**

Step 3: Start Chatting

Type your first message and press Enter:

```
You: Hello! Can you explain what Rust is in simple terms?

Assistant: Hello! Rust is a modern programming language that's designed to be both
fast and safe. Here are the key things that make Rust special:

**Speed**: Rust programs run as fast as C and C++ programs because it compiles
directly to machine code.

**Safety**: Unlike C/C++, Rust prevents common programming errors like accessing
invalid memory or data races in concurrent programs.

**No Garbage Collector**: Rust manages memory automatically without needing a
garbage collector, which keeps programs fast and predictable.

**Growing Ecosystem**: It's increasingly used for web backends, system programming,
blockchain, and even WebAssembly applications.

Think of Rust as giving you the performance of low-level languages like C, but
with the safety and ergonomics of higher-level languages like Python or Java.

You:
```

Congratulations! 🎉 You've successfully started your first conversation with Perspt.

1.4.4 Basic Commands

While chatting, you can use these keyboard shortcuts:

Shortcut	Action
Enter	Send your message
Ctrl+C	Exit Perspt
↑/↓ Arrow Keys	Scroll through chat history
Page Up/Down	Scroll chat quickly
Ctrl+L	Clear the screen

1.4.5 Switching Models

You can easily switch between different AI models and providers:

OpenAI Models

```
# Use GPT-4
perspt --model-name gpt-4

# Use GPT-4 Turbo
perspt --model-name gpt-4-turbo-preview

# Use GPT-4o Mini (recommended for most use cases)
perspt --model-name gpt-4o-mini

# Use latest GPT-4.1
perspt --model-name gpt-4.1
```

Other Providers

```
# Use Anthropic Claude
perspt --provider-type anthropic --model-name claude-3-sonnet-20240229

# Use Google Gemini
perspt --provider-type google --model-name gemini-pro

# Use Ollama (Local)
perspt --provider-type ollama --model-name llama3.2
```

List Available Models

```
# See all available models for your provider
perspt --list-models
```

1.4.6 Basic Configuration

For frequent use, create a configuration file to set your preferences:

Create Config File

```
# Create a config.json file  
touch config.json
```

Add your configuration:

```
{  
  "api_key": "your-api-key-here",  
  "default_model": "gpt-4o-mini",  
  "default_provider": "openai",  
  "provider_type": "openai"  
}
```

Use Config File

```
# Use your configuration file  
perspt --config config.json  
  
# Or place config.json in the same directory as perspt  
perspt
```

1.4.7 Common First-Time Issues

Issue: “API key not found”

Solution: Make sure your API key is properly set:

```
# Check if the key is set  
echo $OPENAI_API_KEY  
  
# If empty, set it again  
export OPENAI_API_KEY="sk-your-key-here"
```

Issue: “Model not available”

Solution: Check available models for your provider:

```
# List available models  
perspt --list-models  
  
# Use a specific model that's available  
perspt --model-name gpt-4o-mini
```

Issue: “Network connection failed”

Solution: Check your internet connection and API key permissions:


```
# Test with a simple curl command  
curl -H "Authorization: Bearer $OPENAI_API_KEY" \  
  "https://api.openai.com/v1/models"
```


Issue: Terminal display problems**Solution:** Ensure your terminal supports modern features:


```
# Try a different terminal emulator if needed
# Update your terminal if it's very old
# Check terminal size
echo $COLUMNS x $LINES
```


1.4.8 Next Steps

Now that you have Perspt running, explore these topics:

 **User Guide** Learn about advanced features, keyboard shortcuts, and productivity tips.

User Guide  **Configuration** Customize Perspt to fit your workflow perfectly.

Configuration Guide  **Multiple Providers** Set up and switch between different AI providers.

AI Providers  **Advanced Features** Discover powerful features like input queuing and markdown rendering.

Advanced Features

1.4.9 Example Conversations

Here are some example conversations to inspire your Perspt usage:

Code Review

You: Can you review this Rust function?

```
```rust
fn calculate_fibonacci(n: u32) -> u32 {
 if n <= 1 {
 return n;
 }
 calculate_fibonacci(n - 1) + calculate_fibonacci(n - 2)
}
```
```

Assistant: I see you've implemented a recursive Fibonacci function. Here's my review:

****Issues:****

- **Performance**:** This has exponential time complexity $O(2^n)$ due to repeated calculations
- **Stack overflow**:** Large values of `n` will cause stack overflow
- **Type limitations**:** `u32` limits the range of Fibonacci numbers you can calculate

****Improved version:****

```
```rust
fn fibonacci_iterative(n: u32) -> u64 {
 if n <= 1 {
 return n as u64;
 }

 let (mut a, mut b) = (0u64, 1u64);
```

(continues on next page)

(continued from previous page)

```

 for _ in 2..n {
 let temp = a + b;
 a = b;
 b = temp;
 }
 b
}
...

```

This iterative version is  $O(n)$  time and  $O(1)$  space, much more efficient!

### Technical Explanation

You: Explain how async/await works in Rust

Assistant: Async/await in Rust is a powerful feature for handling concurrent operations. Here's how it works:

#### ## The Basics

**Async Functions**: Return a `Future` instead of executing immediately

```

```rust
async fn fetch_data() -> String {
    // This doesn't run until awaited
    "data".to_string()
}
...

```

Await: Actually executes the future

```

```rust
let result = fetch_data().await;
...

```

#### ## Under the Hood

1. **State Machine**: The compiler transforms async functions into state machines
2. **Zero-cost**: No heap allocations for simple futures
3. **Cooperative**: Tasks yield control at await points

#### ## Key Concepts

- **Futures are lazy**: They don't do anything until polled
- **Executors**: Runtime systems (like Tokio) that drive futures to completion
- **Non-blocking**: Awaiting doesn't block the thread

This makes Rust excellent for high-performance concurrent applications!

## 1.4.10 Troubleshooting

### No Provider Detected

If you see an error message like this when launching Perspt:

```
❑ No LLM provider configured!
```

To get started, either:

1. Set an environment variable for a supported provider:

- OPENAI\_API\_KEY=sk-your-key
- ANTHROPIC\_API\_KEY=sk-ant-your-key
- GEMINI\_API\_KEY=your-key
- # ... (shows all supported providers)

2. Use command line arguments:

```
perspt --provider openai --api-key sk-your-key
```

**Solution:** Set at least one API key environment variable:

```
Quick fix - set any supported provider
export OPENAI_API_KEY="sk-your-actual-key"
perspt # Should now auto-detect and start
```

### Provider Priority

If you have multiple API keys set and want to use a specific provider:

```
Override automatic detection
perspt --provider anthropic # Forces Anthropic even if OpenAI key is set

Or unset other providers temporarily
unset OPENAI_API_KEY
export ANTHROPIC_API_KEY="your-key"
perspt # Now auto-detects Anthropic
```

### Connection Issues

If Perspt detects your provider but can't connect:

1. **Check your API key:** Ensure it's valid and has sufficient credits
2. **Test your connection:** Try a simple curl request to the provider's API
3. **Check firewall:** Ensure your network allows HTTPS connections
4. **Try Ollama:** For offline usage, install Ollama for local models

```
Test OpenAI connection
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
 https://api.openai.com/v1/models
```

## 1.4.11 Tips for Success

1. **Start Simple:** Begin with basic conversations before exploring advanced features
2. **Experiment:** Try different models and providers to find what works best for your use case
3. **Use Configuration:** Set up a config file for your most common settings
4. **Join the Community:** Connect with other Perspt users for tips and support

5. **Stay Updated:** Check for updates regularly to get new features and improvements

#### ➡ See also

- *Installation Guide* - Detailed installation instructions
- *Configuration Guide* - Complete configuration guide
- *Basic Usage* - Everyday usage patterns
- *Troubleshooting* - Common issues and solutions


## 1.5 Tutorials

Step-by-step guides to master Perspt.

- **First Chat** Your first conversation with an LLM.

*First Chat*  **Agent Mode** Autonomous code generation with SRBN.

*Agent Mode Tutorial* □ **Local Models** Use Ollama for offline AI.

*Local Models with Ollama*  **Custom Workflows** Build automated pipelines.

*Custom Workflows*

### 1.5.1 Learning Path

Order	Tutorial	Outcome
1	<i>First Chat</i>	Understand basic TUI interaction
2	<i>Local Models with Ollama</i>	Set up Ollama for privacy
3	<i>Agent Mode Tutorial</i>	Master autonomous coding
4	<i>Custom Workflows</i>	Automate complex tasks

#### First Chat

Your first conversation with an LLM using Perspt.

#### Prerequisites

- Perspt installed (see *Quick Start*)
- An API key for any provider

#### Step 1: Set Your API Key

Choose your provider and set the environment variable:

##### OpenAI

```
export OPENAI_API_KEY="sk-..."
```



### Anthropic

```
export ANTHROPIC_API_KEY="sk-ant-..."
```

### Google

```
export GEMINI_API_KEY="..."
```

### Ollama (Local)

```
No key needed, just ensure Ollama is running
ollama serve
```

## Step 2: Launch Perspt

```
perspt
```

Or with a specific model:

```
perspt chat --model gpt-5.2
```

## Step 3: The TUI Interface

You'll see the Perspt TUI:

```
Perspt v0.5.0 - gpt-5.2 Tokens: 0
Welcome to Perspt!
Your Terminal's Window to the AI World
> Type your message here...
```

## Step 4: Send a Message

Type your message and press **Enter**:

```
> What is the capital of France?
```

The response will stream in real-time with markdown rendering.

## Step 5: Continue the Conversation

Keep chatting! The conversation history is maintained:

```
> And what's the population?
```

The population of Paris is approximately 2.1 million in the city proper, and about 12 million in the metropolitan area.

### Step 6: Save Your Conversation

Use the `/save` command:

```
> /save my_chat.md
```

Or with automatic timestamp:

```
> /save
```

### Step 7: Exit

Press **Esc** or **Ctrl+C** to exit cleanly.

### Key Bindings Reference

Key	Action
<b>Enter</b>	Send message
<b>Esc</b>	Exit application
↑/↓	Scroll chat history
<b>Page Up/Down</b>	Fast scroll
<b>Ctrl+C</b>	Force exit

### Tips

1. **Markdown works:** Use code, **bold**, and lists in your prompts
2. **Long responses:** Scroll up to see earlier content
3. **Token tracking:** Watch the token counter in the header
4. **Model switching:** Use `perspt chat --model <name>` for different models

### Next Steps

- *Local Models with Ollama* — Use Ollama for offline AI
- *Agent Mode Tutorial* — Try autonomous code generation
- *Configuration* — Customize your setup

### Agent Mode Tutorial

Master autonomous code generation with SRBN.

### Overview

Agent Mode lets Perspt autonomously write, test, and verify code using the **Stabilized Recursive Barrier Network (SRBN)** engine.

## Prerequisites

- Perspt v0.5.0+ installed
- API key for a capable model (GPT-5.2, Claude Opus 4.5 recommended)
- Python 3.9+ (for LSP integration)

## Basic Usage

```
Simple task
perspt agent "Create a Python calculator"

With workspace
perspt agent -w ./my-project "Add unit tests"

Auto-approve all
perspt agent -y "Refactor error handling"
```

## Step-by-Step Example

Let's create a Python calculator:

### Step 1: Start the Agent

```
mkdir calculator-demo && cd calculator-demo
perspt agent "Create a Python calculator with add, subtract, multiply, divide operations.
↳ Include type hints and a comprehensive test suite."
```

### Step 2: Watch the SRBN Loop

The agent will:

1. **Sheafify**: Decompose into subtasks

```
{
 "nodes": [
 {"id": 1, "description": "Create Calculator class"},
 {"id": 2, "description": "Add arithmetic methods"},
 {"id": 3, "description": "Write unit tests"}
]
}
```

2. **Speculate**: Generate code for each node
3. **Verify**: Check with LSP and tests

```
V(x) = 1.0·V_syn + 0.5·V_str + 2.0·V_log
V_syn = 0 (no LSP errors)
V_str = 0.1 (clean structure)
V_log = 0 (all tests pass)
V(x) = 0.05 < ε (stable!)
```

4. **Commit**: Record in ledger

Step 3: Review Changes

When prompted, review the generated code:

```
Review Changes

+ calculator.py (new file, 45 lines)
+ test_calculator.py (new file, 62 lines)

[y] Approve [n] Reject [d] View Diff
```

Step 4: Check Results

```
View generated files
ls -la
calculator.py
test_calculator.py

Run tests
python -m pytest test_calculator.py -v
```

Model Tier Configuration

Use specialized models for different SRBN phases:

```
perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model claude-opus-4.5 \
 --verifier-model gemini-3-pro \
 --speculator-model gemini-3-flash \
 "Build a REST API"
```

Tier	Purpose	Recommendation
Architect	Task decomposition	Deep reasoning (GPT-5.2)
Actuator	Code generation	Strong coding (Claude)
Verifier	Stability check	Fast analysis (Gemini Pro)
Speculator	Branch prediction	Ultra-fast (Gemini Flash)

Energy Tuning

Customize the Lyapunov energy weights:

```
Prioritize test passing (higher γ)
perspt agent --energy-weights "1.0,0.5,3.0" "Add tests"

Prioritize type safety (higher α)
perspt agent --energy-weights "2.0,0.5,1.0" "Add type hints"
```

## Execution Modes

Mode	Behavior
cautious	Prompt for approval on every change
balanced	Prompt when complexity > K (default)
yolo	Auto-approve everything (dangerous!)

```
perspt agent --mode cautious "Modify database schema"
```

## Complexity Threshold

Control when to prompt for approval:

```
Approve up to 3 files without prompting
perspt agent -k 3 "Refactor module"
```

```
Always prompt (k=0)
perspt agent -k 0 "Any task"
```

## Cost and Step Limits

```
Maximum $5 cost
perspt agent --max-cost 5.0 "Large refactor"
```

```
Maximum 10 iterations
perspt agent --max-steps 10 "Iterative improvement"
```

## Managing Sessions

```
Check status
perspt status
```

```
Abort current
perspt abort
```

```
Resume interrupted
perspt resume
```

## Change Tracking

```
View history
perspt ledger --recent
```

```
Rollback
perspt ledger --rollback abc123
```

```
Statistics
perspt ledger --stats
```

## Best Practices

1. **Start small:** Test with simple tasks first
2. **Use workspace:** Always specify `-w` for clarity
3. **Set limits:** Use `--max-cost` and `--max-steps`
4. **Review carefully:** Check diffs before approving
5. **Use tiers:** Match models to task requirements
6. **Track changes:** Use `perspt ledger` regularly

## Troubleshooting

### Agent stuck in retry loop:

- Check LSP is working: `ty check file.py`
- Lower stability threshold: `--stability-threshold 0.5`
- Reduce energy weights for less strict verification

### High energy despite clean code:

- Check test failures: `pytest -v`
- Review LSP diagnostics
- Adjust weights: `--energy-weights "0.5,0.5,1.0"`



## See Also

- *SRBN Architecture* - SRBN details
- *Agent Options* - Full CLI reference
- *perspt-agent API* - API documentation

## Local Models with Ollama

Run AI locally with no API keys or internet required.

### Why Local Models?

 <b>Privacy</b>	All data stays on your machine
<input type="checkbox"/> <b>Cost</b>	No API fees or usage limits
 <b>Offline</b>	Works without internet
<input type="checkbox"/> <b>Experimentation</b>	Test models freely

## Install Ollama

### macOS

```
brew install ollama
```

### Linux

```
curl -fsSL https://ollama.ai/install.sh | sh
```

Windows

Download from [ollama.ai](https://ollama.ai)

Start Ollama

```
ollama serve
```

Pull a Model

```
Recommended for coding
ollama pull llama3.2 # General purpose
ollama pull codellama # Code-focused
ollama pull deepseek-coder # Coding specialist
ollama pull qwen2.5-coder # Code completion
```

Use with Perspt

```
Chat mode
perspt chat --model llama3.2

Agent mode
perspt agent --model codellama "Create a Python script"
```


Model Recommendations

Task	Model	Notes
General chat	llama3.2	Best all-around
Code generation	codellama:13b	Good for agent mode
Code completion	qwen2.5-coder	Fast, accurate
Reasoning	deepseek-coder:33b	Complex tasks

Agent Mode with Local Models

Local models can power SRBN, but with considerations:

```
Use local for all tiers
perspt agent \
 --architect-model deepseek-coder:33b \
 --actuator-model codellama:13b \
 --verifier-model llama3.2 \
 --speculator-model llama3.2 \
 "Create a web scraper"
```

 **Performance Note**

Local models are slower than cloud APIs. For complex agent tasks, consider using a capable cloud model for the Architect tier.

## Hybrid Approach

Use cloud for planning, local for execution:

```
perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model codellama:13b \
 "Build an API"
```

## GPU Acceleration

For faster inference:

```
Check GPU usage
ollama ps

Most models auto-detect GPU
For manual control:
OLLAMA_GPU_LAYERS=35 ollama serve
```

## Troubleshooting

**Model not found:**

```
ollama list # Show installed models
ollama pull <model> # Install missing model
```

**Slow performance:**

- Use smaller models (7B instead of 13B)
- Ensure GPU is being used
- Increase OLLAMA\_NUM\_PARALLEL

**Connection refused:**

```
Ensure Ollama is running
ollama serve

Check port (default 11434)
curl http://localhost:11434/api/tags
```

## See Also

- [Ollama Documentation](#)
- *First Chat* - Basic usage
- *Agent Mode Tutorial* - Autonomous coding

## Custom Workflows

Build automated pipelines with Perspt.



## Overview

Perspt can be integrated into automated workflows for:

- CI/CD code generation
- Batch processing
- Scripted interactions
- Test automation

## Scripting with Agent Mode

Use agent mode in scripts:

```
#!/bin/bash
generate_tests.sh

for file in src/*.py; do
 perspt agent -y -w . "Add unit tests for $file"
done
```

## Batch Code Generation

Process a list of tasks:

```
#!/bin/bash
batch_tasks.sh

TASKS=(
 "Add type hints to utils.py"
 "Create docstrings for api.py"
 "Add error handling to db.py"
)

for task in "${TASKS[@]}; do
 echo "Processing: $task"
 perspt agent -y --max-cost 1.0 "$task"
done
```

## CI/CD Integration

GitHub Actions example:

```
.github/workflows/code-review.yml
name: AI Code Review
on: [pull_request]

jobs:
 review:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v4

 - name: Install Perspt
```

(continues on next page)

(continued from previous page)

```

run: cargo install perspt

- name: Run AI Review
 env:
 OPENAI_API_KEY: ${ secrets.OPENAI_API_KEY }
 run: |
 perspt agent -y "Review code changes and suggest improvements"

```

## Programmatic API

Use perspt-agent crate directly:

```

use perspt_agent::{SRBNOrchestrator, OrchestratorOptions};
use perspt_core::GenAIProvider;
use std::sync::Arc;

#[tokio::main]
async fn main() -> anyhow::Result<()> {
 let provider = Arc::new(GenAIProvider::new()?);

 let options = OrchestratorOptions {
 architect_model: Some("gpt-5.2".to_string()),
 actuator_model: Some("claude-opus-4.5".to_string()),
 ..Default::default()
 };

 let mut orchestrator = SRBNOrchestrator::new(
 provider,
 ".".into(),
 options,
).await?;

 let result = orchestrator.execute("Add unit tests").await?;
 println!("Result: {:?}", result);

 Ok(())
}

```

## Ledger Automation

Automate rollbacks on failure:

```

#!/bin/bash
safe_agent.sh

Store current state
BEFORE=$(perspt ledger --recent | head -1 | cut -d' ' -f1)

Run agent
perspt agent -y "$1"

```

(continues on next page)

(continued from previous page)

```
Run tests
if ! python -m pytest; then
 echo "Tests failed, rolling back..."
 perspt ledger --rollback "$BEFORE"
 exit 1
fi

echo "Success!"
```

## Policy Automation

Create project-specific rules:

```
Initialize with rules
perspt init --rules

Edit .perspt/rules.star
cat > .perspt/rules.star << 'EOF'
Allow read operations
allow("cat *")
allow("ls *")

Prompt for writes
prompt("rm *", reason="File deletion")


Deny dangerous
deny("rm -rf *")
EOF
```


## See Also


- *Agent Mode Tutorial* - Agent fundamentals
- *Configuration* - Project config
- *perspt-agent API* - Programmatic API

## 1.6 User Guide

Comprehensive guide for Perspt users.

 **Basic Usage** Day-to-day operations and workflows.

*Basic Usage*  **Advanced Features** Power user capabilities.

*Advanced Features*  **Agent Mode** Autonomous code generation with SRBN.

*Agent Mode* ☐ **Providers** Working with different LLM providers.

*AI Providers* ☐ **Troubleshooting** Common issues and solutions.

*Troubleshooting*

## 1.6.1 Contents

### Basic Usage

Day-to-day operations with Perspt.

### Starting Perspt

```
Default: starts chat TUI
perspt

With specific model
perspt chat --model gpt-5.2

From anywhere
export PATH="$PATH:/path/to/perspt/target/release"
perspt
```

### Chat Interface

Perspt v0.5.0 – gpt-5.2

Tokens: 1,234 |

User: What is recursion?

Assistant: Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller subproblems.

```
```python
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)
```
```

> \_

### Sending Messages

1. Type your message in the input area
2. Press **Enter** to send
3. Watch the streamed response with markdown rendering

## Key Actions

|                       |                        |
|-----------------------|------------------------|
| <b>Scroll up/down</b> | View previous messages |
| <b>Esc</b>            | Exit Perspt            |
| <b>/save</b>          | Save conversation      |
| <b>/clear</b>         | Clear history          |

## Saving Conversations

```
> /save
Saved to: conversation_2024-12-23_15-30-00.md

> /save my_chat.md
Saved to: my_chat.md
```

## Switching Models

```
In a new session
perspt chat --model claude-opus-4.5

In chat (if supported)
> /model gemini-3-flash
```

## Working with Code

Perspt renders code blocks with syntax highlighting:

```
> Write a Python function to reverse a string

Here's a concise solution:

```python
def reverse_string(s: str) -> str:
    return s[::-1]

# Usage
print(reverse_string("hello")) # "olleh"
```
```

## Multi-turn Conversations

The chat maintains context:

```
> Write a Calculator class

```python
class Calculator:
    def add(self, a, b): return a + b
    def subtract(self, a, b): return a - b
```
```

(continues on next page)

(continued from previous page)

```
> Add multiply and divide methods

```python
class Calculator:
    def add(self, a, b): return a + b
    def subtract(self, a, b): return a - b
    def multiply(self, a, b): return a * b
    def divide(self, a, b):
        if b == 0: raise ValueError("Cannot divide by zero")
        return a / b
```
```

### Token Usage

The header shows cumulative token usage:

- **Input tokens:** Your prompts
- **Output tokens:** AI responses
- **Total:** Running sum for cost estimation

### Exit

Press **Esc** or **Ctrl+C** for a clean exit.

### See Also

- *Advanced Features* - Power user features
- *Agent Mode* - Autonomous coding
- *Configuration* - Configuration

### Advanced Features

Power user capabilities in Perspt.

### Model Tier Configuration

Use specialized models for each SRBN phase:

```
perspt agent \
--architect-model gpt-5.2 \
--actuator-model claude-opus-4.5 \
--verifier-model gemini-3-pro \
--speculator-model gemini-3-flash \
"Build API"
```

### Energy Tuning

Customize Lyapunov energy weights:

$$V(x) = \alpha \cdot V_{syn} + \beta \cdot V_{str} + \gamma \cdot V_{log}$$

```
Prioritize tests (raise γ)
perspt agent --energy-weights "1.0,0.5,3.0" "Add tests"

Prioritize type safety (raise α)
perspt agent --energy-weights "2.0,0.5,1.0" "Add type hints"
```

### Stability Threshold

Control convergence sensitivity:

```
Stricter (production)
perspt agent --stability-threshold 0.05 "Critical fix"

Lenient (prototyping)
perspt agent --stability-threshold 0.5 "Quick draft"
```

### Execution Modes

| Mode     | Behavior                                                                               |
|----------|----------------------------------------------------------------------------------------|
| cautious | Prompt for every change                                                                |
| balanced | Prompt when complexity > K (default)                                                   |
| yolo     | Auto-approve everything ( <input type="checkbox"/> <input type="checkbox"/> dangerous) |

### Cost and Step Limits

```
Set budget
perspt agent --max-cost 5.0 "Large refactor"

Limit iterations
perspt agent --max-steps 10 "Iterative task"
```

### Merkle Ledger

Track and rollback changes:

```
View history
perspt ledger --recent

Rollback
perspt ledger --rollback abc123

Statistics
perspt ledger --stats
```

### Policy Rules

Create custom Starlark rules:

```
.perspt/rules.star
allow("cat *")
prompt("rm *", reason="File deletion")
deny("rm -rf /")
```

## Project Memory

Use PERSPT.md for project context:

```
My Project

Tech Stack
- Python 3.11
- FastAPI
- PostgreSQL

Conventions
- Type hints everywhere
- 100% test coverage
```

## Session Management

```
Check status
perspt status

Abort
perspt abort

Resume
perspt resume
```

## See Also

- *Agent Options* - Full CLI reference
- *SRBN Architecture* - SRBN details

## Agent Mode

Autonomous code generation with SRBN.

## What is Agent Mode?

Agent Mode uses the **Stabilized Recursive Barrier Network (SRBN)** to autonomously:

1. Decompose tasks into subtasks
2. Generate code for each subtask
3. Verify with LSP and tests
4. Commit stable changes to the Merkle ledger



Quick Start

```
perspt agent "Create a Python calculator"
```

How It Works

Model Tiers

| Tier       | Purpose            | Example                           |
|------------|--------------------|-----------------------------------|
| Architect  | Task decomposition | --architect-model gpt-5.2         |
| Actuator   | Code generation    | --actuator-model claude-opus-4.5  |
| Verifier   | Stability check    | --verifier-model gemini-3-pro     |
| Speculator | Fast lookahead     | --speculator-model gemini-3-flash |

Common Commands

```
Basic
perspt agent "Create module"

With workspace
perspt agent -w ./project "Add tests"

Auto-approve
perspt agent -y "Refactor"

Production-grade
perspt agent \
 --architect-model gpt-5.2 \
 --stability-threshold 0.05 \
 --max-cost 10.0 \
 "Implement auth"
```

Review Process

When changes need approval:

Review Changes

+ main.py (new)  
+ tests/test\_main.py (new)

[y] Approve [n] Reject

Session Control

```
perspt status # Check progress
perspt abort # Cancel
perspt resume # Resume
```

## See Also

- *Agent Mode Tutorial* - Full tutorial
- *SRBN Architecture* - Technical details
- *Agent Options* - CLI reference

## AI Providers

This comprehensive guide covers all supported AI providers in Perspt powered by the modern genai crate (v0.3.5), their latest capabilities, configuration options, and best practices for optimal performance.

## Overview

Perspt leverages the unified genai crate to provide seamless access to multiple AI providers with consistent APIs and enhanced features:

OpenAI Latest GPT models including reasoning models (o1-series), GPT-4.1, and optimized variants

Anthropic Claude 3.5 family with constitutional AI and safety-focused design

Google AI Gemini 2.5 Pro and multimodal capabilities with large context windows

Groq Ultra-fast inference with Llama and Mixtral models

Cohere Command R+ models optimized for business and RAG applications

XAI Grok models with real-time web access and humor

Ollama Local model hosting with privacy and offline capabilities

XAI Grok models for advanced reasoning and conversation

## OpenAI

OpenAI provides cutting-edge language models including the latest reasoning capabilities through the genai crate integration.

## Supported Models

| Model         | Context Length | Best For                                | Notes                                         |
|---------------|----------------|-----------------------------------------|-----------------------------------------------|
| gpt-4.1       | 128K tokens    | Enhanced reasoning, latest capabilities | Most advanced GPT-4 variant (2025)            |
| o1-preview    | 128K tokens    | Complex reasoning, problem solving      | Advanced reasoning with step-by-step thinking |
| o1-mini       | 128K tokens    | Fast reasoning, coding tasks            | Efficient reasoning model                     |
| o3-mini       | 128K tokens    | Latest reasoning capabilities           | Newest reasoning model (2025)                 |
| gpt-4o        | 128K tokens    | Multimodal, fast performance            | Optimized for speed and quality               |
| gpt-4o-mini   | 128K tokens    | Fast, cost-effective (default)          | Efficient version of GPT-4o                   |
| gpt-4-turbo   | 128K tokens    | Complex reasoning, analysis             | Previous generation flagship                  |
| gpt-3.5-turbo | 16K tokens     | Fast, cost-effective                    | Good for simple tasks                         |

## Configuration

Basic OpenAI configuration with genai crate:

```
{
 "provider_type": "openai",
 "api_key": "sk-your-openai-api-key",
 "default_model": "gpt-4o-mini",
 "providers": {
 "openai": "https://api.openai.com/v1"
 }
}
```

## CLI Usage

```
Use latest reasoning model
perspt --provider-type openai --model o1-mini

Use fastest model (default)
perspt --provider-type openai --model gpt-4o-mini

List all available OpenAI models
perspt --provider-type openai --list-models
```

## Reasoning Model Features

O1-series models provide enhanced reasoning with visual feedback:

```
> Solve this logic puzzle: There are 5 houses in a row...

[Reasoning...] Let me work through this step by step:
1. Setting up the constraints...
2. Analyzing the color clues...
3. Cross-referencing with pet information...
[Streaming...] Based on my analysis, here's the solution...
```

## Environment Variables

```
export OPENAI_API_KEY="sk-your-key-here"
export OPENAI_ORG_ID="org-your-org-id" # Optional
```

## Anthropic (Claude)

Anthropic's Claude models excel at safety, reasoning, and nuanced understanding through constitutional AI principles.

Supported Models

| Model                      | Context Length | Best For                             | Notes                     |
|----------------------------|----------------|--------------------------------------|---------------------------|
| claude-3-5-sonnet-20241022 | 200K tokens    | Balanced performance, latest version | Recommended default       |
| claude-3-5-sonnet-20240620 | 200K tokens    | Previous Sonnet version              | Stable and reliable       |
| claude-3-5-haiku-20241022  | 200K tokens    | Fast responses, cost-effective       | Good for simple tasks     |
| claude-3-opus-20240229     | 200K tokens    | Most capable, complex reasoning      | Highest quality responses |

Configuration

```
{
 "provider_type": "anthropic",
 "api_key": "sk-ant-your-anthropic-key",
 "default_model": "claude-3-5-sonnet-20241022",
 "providers": {
 "anthropic": "https://api.anthropic.com"
 }
}
```

CLI Usage

```
Use latest Claude model
perspt --provider-type anthropic --model claude-3-5-sonnet-20241022

Use fastest Claude model
perspt --provider-type anthropic --model claude-3-5-haiku-20241022

List available Anthropic models
perspt --provider-type anthropic --list-models
```

Environment Variables

```
export ANTHROPIC_API_KEY="sk-ant-your-key-here"
```

Google AI (Gemini)

Google’s Gemini models offer multimodal capabilities and large context windows with competitive performance.

## Supported Models

| Model                | Context Length | Best For                          | Notes                            |
|----------------------|----------------|-----------------------------------|----------------------------------|
| gemini-2.0-flash-exp | 1M tokens      | Latest experimental model         | Cutting-edge capabilities (2025) |
| gemini-1.5-pro       | 2M tokens      | Large documents, complex analysis | Largest context window           |
| gemini-1.5-flash     | 1M tokens      | Fast responses, good balance      | Recommended default              |
| gemini-pro           | 32K tokens     | General purpose tasks             | Stable and reliable              |

## Configuration

```
{
 "provider_type": "google",
 "api_key": "your-google-api-key",
 "default_model": "gemini-1.5-flash",
 "providers": {
 "google": "https://generativelanguage.googleapis.com"
 }
}
```

## CLI Usage

```
Use latest Gemini model
perspt --provider-type google --model gemini-2.0-flash-exp

Use model with largest context
perspt --provider-type google --model gemini-1.5-pro

List available Google models
perspt --provider-type google --list-models
```

## Environment Variables

```
export GOOGLE_API_KEY="your-key-here"
or
export GEMINI_API_KEY="your-key-here"
 "User-Agent": "Perspt/1.0"
}
```

## Best Practices

1. **Model Selection:** - Use gpt-4-turbo for complex reasoning tasks - Use gpt-3.5-turbo for simple queries to save costs - Use gpt-4-vision-preview when working with images
2. **Token Management:** - Monitor usage with longer conversations - Use appropriate max\_tokens limits - Consider conversation history truncation
3. **Rate Limits:** - Implement retry logic for rate limit errors - Consider upgrading to higher tier plans for increased limits

## Anthropic (Claude)

Anthropic's Claude models are known for their helpfulness, harmlessness, and honesty.

### Supported Models

| Model                    | Context Length | Best For                          | Notes                      |
|--------------------------|----------------|-----------------------------------|----------------------------|
| claude-3-opus-20240229   | 200K tokens    | Complex reasoning, creative tasks | Most capable Claude model  |
| claude-3-sonnet-20240229 | 200K tokens    | Balanced performance/speed        | Good general-purpose model |
| claude-3-haiku-20240307  | 200K tokens    | Fast responses, simple tasks      | Most cost-effective        |
| claude-2.1               | 200K tokens    | Legacy support                    | Deprecated, use Claude-3   |

### Configuration

Basic Anthropic configuration:

```
{
 "provider": "anthropic",
 "api_key": "your-anthropic-api-key",
 "model": "claude-3-opus-20240229",
 "base_url": "https://api.anthropic.com",
 "version": "2023-06-01",
 "max_tokens": 4000,
 "temperature": 0.7,
 "top_p": 1.0,
 "top_k": 40,
 "stop_sequences": ["\\n\\nHuman:", "\\n\\nAssistant:"]
}
```

### Advanced Configuration

System Messages:

```
{
 "provider": "anthropic",
 "model": "claude-3-opus-20240229",
 "system_message": "You are a helpful assistant specialized in software development.↵↵Provide detailed, accurate responses with code examples when appropriate."
}
```

Content Filtering:

```
{
 "provider": "anthropic",
 "content_filtering": {
 "enabled": true,
 "strictness": "moderate"
 }
}
```

## Best Practices

1. **Model Selection:** - Use `claude-3-opus` for complex analysis and creative work - Use `claude-3-sonnet` for balanced general-purpose tasks - Use `claude-3-haiku` for quick questions and simple tasks
2. **Prompt Engineering:** - Claude responds well to clear, structured prompts - Use explicit instructions and examples - Leverage Claude's strong reasoning capabilities
3. **Long Conversations:** - Take advantage of the large context window - Maintain conversation flow without frequent truncation

## Google AI (Gemini)

Google's Gemini models offer strong reasoning and multimodal capabilities.

## Supported Models

| Model                          | Context Length | Best For                        | Notes                         |
|--------------------------------|----------------|---------------------------------|-------------------------------|
| <code>gemini-2.5-pro</code>    | 2M tokens      | Advanced reasoning, analysis    | Latest and most capable       |
| <code>gemini-2.0-flash</code>  | 1M tokens      | Fast, efficient performance     | Optimized for speed           |
| <code>gemini-1.5-pro</code>    | 2M tokens      | Complex reasoning, long context | High-capability model         |
| <code>gemini-1.5-flash</code>  | 1M tokens      | Fast responses, good quality    | Balanced speed and capability |
| <code>gemini-pro</code>        | 32K tokens     | General reasoning               | Legacy model                  |
| <code>gemini-pro-vision</code> | 16K tokens     | Multimodal tasks                | Supports images and text      |

## Configuration

Basic Google AI configuration:

```
{
 "provider": "google",
 "api_key": "your-google-api-key",
 "model": "gemini-pro",
 "base_url": "https://generativelanguage.googleapis.com/v1",
 "safety_settings": {
 "harassment": "BLOCK_MEDIUM_AND_ABOVE",
 "hate_speech": "BLOCK_MEDIUM_AND_ABOVE",
 "sexually_explicit": "BLOCK_MEDIUM_AND_ABOVE",
 "dangerous_content": "BLOCK_MEDIUM_AND_ABOVE"
 },
 "generation_config": {
 "temperature": 0.7,
 "top_p": 1.0,
 "top_k": 40,
 "max_output_tokens": 4000
 }
}
```

## Multimodal Configuration

For image analysis with Gemini Vision:

```
{
 "provider": "google",
 "model": "gemini-pro-vision",
 "multimodal": {
 "enabled": true,
 "supported_formats": ["png", "jpg", "jpeg", "webp", "gif"],
 "max_image_size": "20MB"
 }
}
```

## Best Practices

1. **Safety Settings:** - Configure appropriate safety levels for your use case - Consider more permissive settings for creative tasks
2. **Multimodal Usage:** - Use Gemini Vision for image analysis and understanding - Combine text and images for richer interactions

## Local Models

Perspt supports various local inference solutions for privacy and offline usage.

### Ollama

Configuration for Ollama local models:

```
{
 "provider": "ollama",
 "base_url": "http://localhost:11434",
 "model": "llama2:7b",
 "stream": true,
 "options": {
 "temperature": 0.7,
 "top_p": 0.9,
 "top_k": 40,
 "repeat_penalty": 1.1,
 "seed": -1,
 "num_ctx": 4096
 }
}
```

Popular Ollama Models:

```
Install popular models
ollama pull llama2:7b # General purpose
ollama pull codellama:7b # Code generation
ollama pull mistral:7b # Fast and capable
ollama pull neural-chat:7b # Conversational
```



LM Studio

Configuration for LM Studio:

```
{
 "provider": "lm_studio",
 "base_url": "http://localhost:1234/v1",
 "model": "local-model",
 "stream": true,
 "context_length": 4096,
 "gpu_layers": 35
}
```

OpenAI-Compatible Servers

For other OpenAI-compatible local servers:

```
{
 "provider": "openai_compatible",
 "base_url": "http://localhost:8000/v1",
 "api_key": "not-needed",
 "model": "local-model-name",
 "stream": true
}
```

Provider Comparison

| Provider       | Speed      | Quality   | Cost   | Privacy | Context  | Multi-modal |
|----------------|------------|-----------|--------|---------|----------|-------------|
| OpenAI         | Fast       | Excellent | Medium | Cloud   | 128K     | Yes         |
| Anthropic      | Medium     | Excellent | Medium | Cloud   | 200K     | No          |
| Google AI      | Fast       | Very Good | Low    | Cloud   | 32K      | Yes         |
| Groq           | Ultra-Fast | Excellent | Low    | Cloud   | 32K      | No          |
| Local (Ollama) | Variable   | Good      | Free   | Local   | Variable | Limited     |

Multi-Provider Setup

Configure multiple providers for different use cases:

```
{
 "providers": {
 "primary": {
 "provider": "openai",
 "model": "gpt-4-turbo",
 "api_key": "your-openai-key"
 },
 "coding": {
 "provider": "anthropic",
 "model": "claude-3-opus-20240229",
 "api_key": "your-anthropic-key"
 }
 }
}
```

(continues on next page)

(continued from previous page)

```
{,
 "local": {
 "provider": "ollama",
 "model": "codellama:7b",
 "base_url": "http://localhost:11434"
 }
},
"default_provider": "primary"
}
```

Switch between providers during conversation:

```
> /provider coding
Switched to coding provider (Claude-3 Opus)

> /provider local
Switched to local provider (CodeLlama)
```

## Fallback Configuration

Set up automatic fallbacks:

```
{
 "fallback_chain": [
 {
 "provider": "openai",
 "model": "gpt-4-turbo"
 },
 {
 "provider": "anthropic",
 "model": "claude-3-sonnet-20240229"
 },
 {
 "provider": "ollama",
 "model": "llama2:7b"
 }
],
 "fallback_conditions": [
 "rate_limit_exceeded",
 "api_error",
 "timeout"
]
}
```

## Troubleshooting

### Common Issues

API Key Issues:

```
> /validate-key
Checking API key validity...
```

(continues on next page)

(continued from previous page)

```
✓ OpenAI key: Valid
x Anthropic key: Invalid or expired
```

**Connection Problems:**

```
Test connectivity
curl -H "Authorization: Bearer your-api-key" \\\
 https://api.openai.com/v1/models
```

**Rate Limiting:**

```
{
 "rate_limiting": {
 "requests_per_minute": 60,
 "tokens_per_minute": 40000,
 "retry_strategy": "exponential_backoff",
 "max_retries": 3
 }
}
```

**Performance Optimization****Request Optimization:**

```
{
 "optimization": {
 "batch_requests": true,
 "compress_requests": true,
 "connection_pooling": true,
 "timeout": 30
 }
}
```

**Caching:**

```
{
 "cache": {
 "enabled": true,
 "provider_specific": true,
 "ttl": 3600,
 "max_size": "100MB"
 }
}
```

**Next Steps**

- *Troubleshooting* - Detailed troubleshooting for provider-specific issues
- *Advanced Features* - Advanced features that work with different providers
- *Configuration Guide* - Complete configuration reference
- *Extending Perspt* - Create custom provider integrations

## Groq

Groq provides ultra-fast inference speeds with popular open-source models, optimized for real-time conversations.

### Supported Models

| Model                    | Context Length | Best For                    | Notes                      |
|--------------------------|----------------|-----------------------------|----------------------------|
| llama-3.1-405b-reasoning | 128K tokens    | Complex reasoning, analysis | Largest Llama model        |
| llama-3.1-70b-versatile  | 128K tokens    | Balanced performance        | Good general purpose model |
| llama-3.1-8b-instant     | 128K tokens    | Ultra-fast responses        | Best for speed             |
| mixtral-8x7b-32768       | 32K tokens     | Mixture of experts          | Strong coding capabilities |

### Configuration

```
{
 "provider_type": "groq",
 "api_key": "your-groq-api-key",
 "default_model": "llama-3.1-70b-versatile",
 "providers": {
 "groq": "https://api.groq.com/openai/v1"
 }
}
```

### CLI Usage

```
Ultra-fast responses
perspt --provider-type groq --model llama-3.1-8b-instant

Balanced performance
perspt --provider-type groq --model llama-3.1-70b-versatile
```

### Environment Variables

```
export GROQ_API_KEY="your-key-here"
```

## Cohere

Cohere specializes in enterprise-focused models with strong RAG (Retrieval-Augmented Generation) capabilities.

## Supported Models

| Model          | Context Length | Best For                     | Notes                             |
|----------------|----------------|------------------------------|-----------------------------------|
| command-r-plus | 128K tokens    | RAG, business applications   | Most capable Cohere model         |
| command-r      | 128K tokens    | General purpose, fast        | Good balance of speed and quality |
| command        | 4K tokens      | Simple cost-effective tasks, | Basic model                       |

## Configuration

```
{
 "provider_type": "cohere",
 "api_key": "your-cohere-api-key",
 "default_model": "command-r-plus",
 "providers": {
 "cohere": "https://api.cohere.ai"
 }
}
```

## Environment Variables

```
export COHERE_API_KEY="your-key-here"
```

## XAI (Grok)

XAI's Grok models provide real-time web access and are known for their humor and current knowledge.

## Supported Models

| Model            | Context Length | Best For              | Notes               |
|------------------|----------------|-----------------------|---------------------|
| grok-beta        | 128K tokens    | Current events, humor | Latest Grok model   |
| grok-vision-beta | 128K tokens    | Multimodal analysis   | Image understanding |

## Configuration

```
{
 "provider_type": "xai",
 "api_key": "your-xai-api-key",
 "default_model": "grok-beta",
 "providers": {
 "xai": "https://api.x.ai/v1"
 }
}
```

## Environment Variables

```
export XAI_API_KEY="your-key-here"
```

Ollama (Local Models)

Ollama provides local model hosting for privacy, offline usage, and cost control with the genai crate integration. Perfect for testing, development, and privacy-conscious users.

Supported Models

Popular models available through Ollama:

| Model        | Size | RAM Required | Best Use Cases                             |
|--------------|------|--------------|--------------------------------------------|
| llama3.2     | 3B   | ~4GB         | General chat, quick responses, testing     |
| llama3.1:8b  | 8B   | ~8GB         | Better reasoning, longer conversations     |
| llama3.1:70b | 70B  | ~40GB        | Complex reasoning, professional tasks      |
| codellama    | 7B   | ~7GB         | Code generation, debugging, technical docs |
| mistral      | 7B   | ~7GB         | Balanced performance, multilingual         |
| phi3         | 3.8B | ~4GB         | Efficient, resource-constrained systems    |
| qwen2.5:7b   | 7B   | ~7GB         | Strong reasoning, mathematics              |

```
Large models (requires significant RAM)
llama3.1:70b # Most capable local model
qwen2.5:72b # Alibaba's flagship model

Medium models (good balance)
llama3.1:8b # Recommended for most users
mistral-nemo:12b # Mistral's latest
codellama # Specialized for coding

Small models (fast, low resource)
llama3.2 # Latest efficient model (default)
phi3 # Microsoft's compact model
qwen2.5:7b # Compact but capable
```

Setup and Configuration

1. Install Ollama:

```
macOS
brew install ollama

Linux
curl -fsSL https://ollama.com/install.sh | sh
```

2. Download Models:

```
Download recommended starter models
ollama pull llama3.2 # General purpose (3B)
ollama pull codellama # Code assistance (7B)
ollama pull mistral # Balanced performance (7B)
```

(continues on next page)

(continued from previous page)

```
Optional: Download larger models if you have RAM
ollama pull llama3.1:8b # Better reasoning (8B)
ollama pull qwen2.5:7b # Strong at math/logic (7B)

Check what's available
ollama list
```

### 3. Start Ollama Service:

```
Start the service (runs on http://localhost:11434)
ollama serve

Or run in background
nohup ollama serve > ollama.log 2>&1 &
```

### 4. Configure Perspt:

```
{
 "provider_type": "ollama",
 "default_model": "llama3.2",
 "providers": {
 "ollama": "http://localhost:11434/v1"
 },
 "api_key": "not-required"
}
```

## CLI Usage

```
Basic usage (no API key needed!)
perspt --provider-type ollama --model llama3.2

Use specific models for different tasks
perspt --provider-type ollama --model codellama # For coding
perspt --provider-type ollama --model mistral # General purpose
perspt --provider-type ollama --model llama3.1:8b # Better reasoning

List installed Ollama models
perspt --provider-type ollama --list-models

Test connection and performance
perspt --provider-type ollama --model llama3.2 --config ollama_config.json
```

### Testing Different Models

```
Quick test with small model
echo "Explain quantum computing in simple terms" | \
perspt --provider-type ollama --model llama3.2

Coding test with Code Llama
echo "Write a Python function to sort a list" | \
perspt --provider-type ollama --model codellama
```

(continues on next page)

(continued from previous page)

```
Reasoning test with larger model
echo "Solve this logic puzzle: ..." | \
perspt --provider-type ollama --model llama3.1:8b
```

## Performance Monitoring

```
Monitor resource usage
htop # Check CPU/Memory while running

Time responses
time perspt --provider-type ollama --model llama3.2

Compare model speeds
for model in llama3.2 mistral codellama; do
 echo "Testing $model..."
 time echo "What is 2+2?" | perspt --provider-type ollama --model $model
done
```

## Benefits of Local Models

- **Privacy:** Data stays on your machine
- **Offline Usage:** No internet required after setup
- **Cost Control:** No per-token charges
- **Customization:** Fine-tune models for specific tasks

## Environment Variables

```
export OLLAMA_HOST="http://localhost:11434"
```

## Troubleshooting

This comprehensive troubleshooting guide helps you diagnose and resolve issues with Perspt's genai crate integration, provider connectivity, and advanced features.

## Quick Diagnostics

Start with these diagnostic commands to check system status:

```
Check provider connectivity and model availability
perspt --provider-type openai --list-models

Validate specific model
perspt --provider-type anthropic --model claude-3-5-sonnet-20241022 --list-models

Test with minimal configuration
perspt --api-key your-key --provider-type openai --model gpt-3.5-turbo
```

## Environment Variable Check

```
Check if API keys are set
echo $OPENAI_API_KEY
echo $ANTHROPIC_API_KEY
```

(continues on next page)



(continued from previous page)

```
echo $GOOGLE_API_KEY

Verify genai crate can access providers
export RUST_LOG=debug
perspt --provider-type openai --list-models
```

## Common Issues

### GenAI Crate Integration Issues

#### Provider Authentication Failures

```
Error: Authentication failed for provider 'openai'
Caused by: Invalid API key
```

#### Solutions:

##### 1. Verify API key format:

```
OpenAI keys start with 'sk-'
echo $OPENAI_API_KEY | head -c 5 # Should show 'sk-'

Anthropic keys start with 'sk-ant-'
echo $ANTHROPIC_API_KEY | head -c 7 # Should show 'sk-ant-'
```

##### 2. Test API key directly:

```
Test OpenAI API key
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
 https://api.openai.com/v1/models

Test Anthropic API key
curl -H "x-api-key: $ANTHROPIC_API_KEY" \
 https://api.anthropic.com/v1/models
```

##### 3. Check API key permissions and billing:

- Ensure API key has model access permissions - Verify account has sufficient credits/billing set up - Check for rate limiting or usage quotas

#### Model Validation Failures

```
Error: Model 'gpt-4.1' not available for provider 'openai'
Available models: gpt-3.5-turbo, gpt-4, gpt-4-turbo...
```

#### Solutions:

##### 1. Check model availability:

```
List all available models for provider
perspt --provider-type openai --list-models

Search for specific model
perspt --provider-type openai --list-models | grep gpt-4
```

##### 2. Use correct model names:

```
Correct model names (case-sensitive)
perspt --provider-type openai --model gpt-4o-mini # ✓ Correct
perspt --provider-type openai --model GPT-40-Mini # ✗ Wrong case
perspt --provider-type openai --model gpt4o-mini # ✗ Missing hyphen
```

3. **Check provider-specific model access:** - Some models require special access (e.g., GPT-4, Claude Opus) - Verify your account tier supports the requested model - Check if model is in beta/preview status

### Streaming Connection Issues

```
Error: Streaming connection interrupted
Caused by: Connection reset by peer
```

#### Solutions:

1. **Network connectivity check:**

```
Test basic connectivity
ping api.openai.com
ping api.anthropic.com

Check for proxy/firewall issues
curl -I https://api.openai.com/v1/models
```

2. **Provider service status:** - Check OpenAI Status: <https://status.openai.com> - Check Anthropic Status: <https://status.anthropic.com> - Check Google AI Status: <https://status.google.com>
3. **Adjust streaming settings:**

```
{
 "provider_type": "openai",
 "default_model": "gpt-4o-mini",
 "stream_timeout": 30,
 "retry_attempts": 3,
 "buffer_size": 1024
}
```

Common syntax errors:

```
{
 "provider": "openai", // ✗ Comments not allowed in JSON
 "api_key": "sk-...", // ✗ Trailing comma
}
```

Correct format:

```
{
 "provider": "openai",
 "api_key": "sk-..."
}
```

2. **Missing Required Fields:**

```
{
 "provider": "openai"
 // ✗ Missing api_key
}
```

**Solution:** Ensure all required fields are present:

```
{
 "provider": "openai",
 "api_key": "your-api-key",
 "model": "gpt-4"
}
```

### Configuration File Not Found

Error: Configuration file not found at ~/.config/perspt/config.json

#### Solutions:

1. Create the configuration directory:

```
mkdir -p ~/.config/perspt
```

2. Create a basic configuration file:

```
cat > ~/.config/perspt/config.json << EOF
{
 "provider": "openai",
 "api_key": "your-api-key",
 "model": "gpt-4"
}
EOF
```

3. Specify a custom configuration path:

```
perspt --config /path/to/your/config.json
```

## API Connection Issues

### Invalid API Key

Error: Authentication failed – Invalid API key

#### Solutions:

1. **Verify API key format:**

```
OpenAI keys start with 'sk-'
Anthropic keys start with 'sk-ant-'
Check your provider's documentation
```

2. **Test API key manually:**

```
OpenAI
curl -H "Authorization: Bearer YOUR_API_KEY" \
 https://api.openai.com/v1/models

Anthropic
curl -H "x-api-key: YOUR_API_KEY" \
 -H "anthropic-version: 2023-06-01" \
 https://api.anthropic.com/v1/messages
```

3. **Check API key permissions:** - Ensure the key has necessary permissions - Check if the key is associated with the correct organization - Verify the key hasn't expired

## Network Connectivity Issues

Error: Failed to connect to API endpoint

### Solutions:

1. **Check internet connectivity:**

```
ping google.com
curl -I https://api.openai.com
```

2. **Verify firewall/proxy settings:**

```
Check if behind corporate firewall
echo $HTTP_PROXY
echo $HTTPS_PROXY
```

3. **Test with different endpoints:**

```
Try different base URLs
curl https://api.openai.com/v1/models
curl https://api.anthropic.com/v1/models
```

4. **Configure proxy if needed:**

```
{
 "provider": "openai",
 "proxy": {
 "http": "http://proxy.company.com:8080",
 "https": "https://proxy.company.com:8080"
 }
}
```

## Rate Limiting

Error: Rate limit exceeded

### Solutions:

1. **Wait and retry:** - Most rate limits reset within minutes - Implement exponential backoff
2. **Check rate limits:**

```
Check OpenAI rate limits
curl -H "Authorization: Bearer YOUR_API_KEY" \\\
https://api.openai.com/v1/usage
```

3. **Optimize requests:**

```
{
 "rate_limiting": {
 "requests_per_minute": 50,
 "delay_between_requests": 1.2,
 "max_retries": 3
 }
}
```

4. **Upgrade API plan:** - Consider higher-tier plans for increased limits - Contact provider support for enterprise limits

## Model and Response Issues

### Model Not Available

Error: Model 'gpt-5' not found

#### Solutions:

1. Check available models:

```
> /list-models
```

2. Verify model name spelling:

```
{
 "model": "gpt-4-turbo", // ✓ Correct
 "model": "gpt-4-turob" // ✗ Typo
}
```

3. Check provider model availability: - Some models may be region-specific - Newer models might not be available to all users

### Slow Responses

#### Causes and solutions:

1. Large context windows:

```
{
 "max_tokens": 1000, // ✓ Reasonable
 "conversation_history_limit": 20 // ✓ Limit history
}
```

2. Network latency:

```
Test latency to provider
ping api.openai.com
```

3. Provider server load: - Check provider status pages - Try different models or regions

### Unexpected Responses

AI responses seem off-topic or inappropriate

#### Solutions:

1. Review system prompt:

```
{
 "system_prompt": "You are a helpful assistant..." // Clear instructions
}
```

2. Adjust model parameters:

```
{
 "temperature": 0.3, // Lower for more focused responses
 "top_p": 0.8, // Reduce randomness
 "frequency_penalty": 0.2 // Reduce repetition
}
```

3. Clear conversation history:

```
> /clear
```

## Local Model Issues

### Ollama Connection Failed

```
Error: Failed to connect to Ollama at localhost:11434
```

#### Solutions:

1. Check if Ollama is running:

```
Start Ollama
ollama serve

Check if running
curl http://localhost:11434/api/tags
```

2. Verify model is installed:

```
ollama list
ollama pull llama2:7b # Install if missing
```

3. Check port configuration:

```
{
 "provider": "ollama",
 "base_url": "http://localhost:11434" // Correct port
}
```

### Insufficient Memory/GPU

```
Error: Out of memory when loading model
```

#### Solutions:

1. Use smaller models:

```
Instead of 13B model, use 7B
ollama pull llama2:7b
ollama pull mistral:7b
```

2. Adjust GPU layers:

```
{
 "provider": "ollama",
 "options": {
 "num_gpu": 0, // Use CPU only
 "num_thread": 4 // Limit CPU threads
 }
}
```

3. Monitor system resources:

```
Check memory usage
htop
nvidia-smi # For GPU usage
```

## Platform-Specific Issues

### macOS Issues

#### Gatekeeper Blocking Execution

```
"perspt" cannot be opened because it is from an unidentified developer
```

##### Solution:

```
sudo xattr -rd com.apple.quarantine /path/to/perspt
```

#### Homebrew Installation Issues

```
Update Homebrew
brew update
brew upgrade

Clear caches
brew cleanup

Reinstall if needed
brew uninstall perspt
brew install perspt
```

### Linux Issues

#### Missing Shared Libraries

```
error while loading shared libraries: libssl.so.1.1
```

##### Solutions:

```
Ubuntu/Debian
sudo apt update
sudo apt install libssl1.1 libssl-dev

Fedora/RHEL
sudo dnf install openssl-libs openssl-devel

Check library dependencies
ldd /path/to/perspt
```

#### Permission Issues

```
Make executable
chmod +x perspt

Install system-wide
sudo cp perspt /usr/local/bin/
```

## Windows Issues

### PowerShell Execution Policy

```
Check current policy
Get-ExecutionPolicy

Set policy to allow local scripts
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

### Windows Defender False Positive

1. Add Perspt to Windows Defender exclusions
2. Download from official sources only
3. Verify file hashes if available

## Advanced Troubleshooting

### Debug Mode

Enable detailed logging:

```
{
 "debug": {
 "enabled": true,
 "log_level": "trace",
 "log_file": "~/.config/perspt/debug.log"
 }
}
```

Run with verbose output:

```
perspt --verbose --debug
```

### Log Analysis

Check log files for detailed error information:

```
View recent logs
tail -f ~/.config/perspt/perspt.log

Search for specific errors
grep -i "error" ~/.config/perspt/perspt.log

Analyze API calls
grep -i "api" ~/.config/perspt/debug.log
```

### Network Debugging

Capture network traffic:

```
Using tcpdump (Linux/macOS)
sudo tcpdump -i any -n host api.openai.com
```

(continues on next page)



(continued from previous page)

```
Using netstat
netstat -an | grep :443
```

**Test with curl:**

```
Test OpenAI API
curl -v -H "Authorization: Bearer YOUR_API_KEY" \
 -H "Content-Type: application/json" \
 -d '{"model": "gpt-4o-mini", "messages": [{"role": "user", "content": "Hello"}]}' \
 https://api.openai.com/v1/chat/completions
```

**Configuration Debugging****Validate configuration:**

```
Check JSON syntax
python -c "import json; print(json.load(open('config.json')))"

Validate with Perspt
perspt --validate-config
```

**Test minimal configuration:**

```
{
 "provider": "openai",
 "api_key": "your-key",
 "model": "gpt-4o-mini"
}
```

**Performance Debugging****Monitor resource usage:**

```
Monitor CPU and memory
top -p $(pgrep perspt)

Monitor disk I/O
iotop -p $(pgrep perspt)
```

**Profile network usage:**

```
Monitor bandwidth usage
netlimit -p $(pgrep perspt)
```

**Recovery Procedures****Reset Configuration**

1. Backup current configuration:

```
cp ~/.config/perspt/config.json ~/.config/perspt/config.json.backup
```

2. Reset to defaults:

```
rm ~/.config/perspt/config.json
perspt --create-config
```

3. **Restore from backup if needed:**

```
cp ~/.config/perspt/config.json.backup ~/.config/perspt/config.json
```

## Clear Cache and Data

```
Clear conversation history
rm -rf ~/.config/perspt/history/

Clear cache
rm -rf ~/.config/perspt/cache/

Clear temporary files
rm -rf /tmp/perspt*
```

## Complete Reinstallation

```
Remove all Perspt data
rm -rf ~/.config/perspt/
rm -rf ~/.local/share/perspt/

Uninstall and reinstall
(method depends on installation method)
```

## Getting Help

### Community Support

- **GitHub Issues:** Report bugs and feature requests
- **Discussions:** Ask questions and share tips
- **Discord/Slack:** Real-time community support

## Reporting Issues

When reporting issues, include:

1. **System information:**

```
perspt --version
uname -a # or systeminfo on Windows
```

2. **Configuration** (sanitized):

```
{
 "provider": "openai",
 "model": "gpt-4",
 "api_key": "sk-***redacted***"
}
```

3. **Error messages** (full text)
4. **Steps to reproduce**

## 5. Expected vs actual behavior

### Professional Support

For enterprise users:

- **Priority support tickets**
- **Direct communication channels**
- **Custom configuration assistance**
- **Integration consulting**

### Provider-Specific Troubleshooting

#### OpenAI Provider Issues

##### Authentication and API Key Problems

```
Error: Invalid API key for OpenAI
Error: Rate limit exceeded for model gpt-4
```

##### Solutions:

###### 1. API Key Validation:

```
Verify OpenAI API key format (should start with 'sk-')
echo $OPENAI_API_KEY | head -c 3 # Should show 'sk-'

Test API key with curl
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
 https://api.openai.com/v1/models
```

###### 2. Rate Limiting Management:

```
Use tier-appropriate models
perspt --provider-type openai --model gpt-3.5-turbo # Lower tier
perspt --provider-type openai --model gpt-4o-mini # Tier 1+
perspt --provider-type openai --model gpt-4 # Tier 3+
```

3. **Quota and Billing Issues:** - Check OpenAI dashboard for usage limits - Verify payment method is valid - Monitor usage to avoid unexpected charges

##### Model Access Issues

```
Error: Model 'o1-preview' not available
Error: Insufficient quota for GPT-4
```

##### Solutions:

###### 1. Model Tier Requirements:

```
Tier 1 models (widely available)
perspt --provider-type openai --model gpt-3.5-turbo
perspt --provider-type openai --model gpt-4o-mini

Tier 2+ models (higher usage requirements)
perspt --provider-type openai --model gpt-4
perspt --provider-type openai --model gpt-4-turbo
```

(continues on next page)

(continued from previous page)

```
Special access models (invitation/waitlist)
perspt --provider-type openai --model o1-preview
perspt --provider-type openai --model o1-mini
```

2. **Reasoning Model Limitations:** - o1 models have special usage patterns - Higher latency expected for reasoning  
- May have stricter rate limits

## Anthropic Provider Issues

### Claude Model Access

```
Error: Model 'claude-3-opus-20240229' not available
Error: Anthropic API key authentication failed
```

#### Solutions:

1. **API Key Format:**

```
Anthropic keys start with 'sk-ant-'
echo $ANTHROPIC_API_KEY | head -c 7 # Should show 'sk-ant-'

Test with curl
curl -H "x-api-key: $ANTHROPIC_API_KEY" \
 -H "anthropic-version: 2023-06-01" \
 https://api.anthropic.com/v1/models
```

2. **Model Availability:**

```
Generally available models
perspt --provider-type anthropic --model claude-3-5-sonnet-20241022
perspt --provider-type anthropic --model claude-3-5-haiku-20241022

Request access for Opus through Anthropic Console
perspt --provider-type anthropic --model claude-3-opus-20240229
```

3. **Rate Limiting:** - Anthropic has strict rate limits for new accounts - Build up usage history for higher limits - Use Haiku model for testing and development

## Google AI (Gemini) Provider Issues

### API Key and Setup Problems

```
Error: Google AI API key not valid
Error: Gemini model access denied
```

#### Solutions:

1. **API Key Configuration:**

```
Get API key from Google AI Studio
export GOOGLE_API_KEY="your-api-key"
Alternative environment variable
export GEMINI_API_KEY="your-api-key"
```

(continues on next page)

(continued from previous page)

```
Test API access
curl -H "Content-Type: application/json" \
 "https://generativelanguage.googleapis.com/v1beta/models?key=$GOOGLE_API_KEY"
```

## 2. Model Selection:

```
Recommended models
perspt --provider-type google --model gemini-1.5-flash # Fast, cost-effective
perspt --provider-type google --model gemini-1.5-pro # Balanced capability
perspt --provider-type google --model gemini-1.5-pro-exp # Experimental features
```

3. **Geographic Restrictions:** - Some Gemini models have geographic limitations - Check Google AI availability in your region - Use VPN if necessary and allowed by Google's terms

## Groq Provider Issues

### Service Availability

```
Error: Groq service temporarily unavailable
Error: Model inference timeout
```

### Solutions:

1. **Service Reliability:** - Groq prioritizes speed over availability - Configure fallback providers for production use - Monitor Groq status page for outages
2. **Model Selection:**

```
Fast inference models
perspt --provider-type groq --model llama-3.1-8b-instant
perspt --provider-type groq --model mixtral-8x7b-32768
perspt --provider-type groq --model gemma-7b-it
```

3. **Timeout Configuration:**

```
{
 "provider_type": "groq",
 "timeout": 30,
 "retry_attempts": 2,
 "fallback_provider": "openai"
}
```

## Cohere Provider Issues

### API Integration Problems

```
Error: Cohere API authentication failed
Error: Model 'command-r-plus' not accessible
```

### Solutions:

1. **API Key Setup:**

```
export COHERE_API_KEY="your-api-key"

Test API access
```

(continues on next page)

(continued from previous page)

```
curl -H "Authorization: Bearer $COHERE_API_KEY" \
https://api.cohere.ai/v1/models
```

## 2. Model Access:

```
Available Cohere models
perspt --provider-type cohere --model command-r
perspt --provider-type cohere --model command-r-plus
perspt --provider-type cohere --model command-light
```

## XAI (Grok) Provider Issues

### Grok Model Access

```
Error: XAI API key invalid
Error: Grok model not available
```

#### Solutions:

##### 1. API Configuration:

```
export XAI_API_KEY="your-api-key"

Check available models
perspt --provider-type xai --list-models
```

##### 2. Model Selection:

```
Available Grok models
perspt --provider-type xai --model grok-beta
```

## Ollama (Local) Provider Issues

### Service Connection Problems

```
Error: Could not connect to Ollama server
Error: Model not found in Ollama
```

#### Solutions:

##### 1. Ollama Service Management:

```
Check if Ollama is running
curl http://localhost:11434/api/tags

Start Ollama service
ollama serve

Start as background service (macOS)
brew services start ollama
```

##### 2. Model Management:

```
List installed models
ollama list
```

(continues on next page)

(continued from previous page)

```
Install popular models
ollama pull llama3.2:8b
ollama pull mistral:7b
ollama pull codellama:7b

Remove unused models to save space
ollama rm unused-model
```

### 3. Resource Optimization:

```
Check system resources
htop
nvidia-smi # For GPU users

Use smaller models for limited resources
ollama pull llama3.2:3b # 3B parameters
ollama pull phi3:mini # Microsoft Phi-3 Mini
```

### 4. Configuration Tuning:

```
{
 "provider_type": "ollama",
 "base_url": "http://localhost:11434",
 "options": {
 "num_gpu": 1, // Number of GPU layers
 "num_thread": 8, // CPU threads
 "num_ctx": 4096, // Context window
 "temperature": 0.7,
 "top_p": 0.9
 }
}
```

## Performance Optimization

### Response Time Optimization

#### Model Selection for Speed

```
Fastest models by provider
perspt --provider-type groq --model llama-3.1-8b-instant # Groq (fastest)
perspt --provider-type openai --model gpt-4o-mini # OpenAI (fast)
perspt --provider-type google --model gemini-1.5-flash # Google (fast)
perspt --provider-type anthropic --model claude-3-5-haiku-20241022 # Anthropic (fast)
```

#### Configuration Tuning

```
{
 "performance": {
 "max_tokens": 1000, // Limit response length
 "stream": true, // Enable streaming
 "timeout": 15, // Shorter timeout
 "parallel_requests": 2, // Multiple requests
 }
}
```

(continues on next page)

(continued from previous page)

```
"cache_responses": true // Cache similar queries
}
}
```

## Memory and Resource Management

### System Resource Monitoring

```
Monitor CPU and memory usage
top -p $(pgrep perspt)

Monitor network usage
iftop -i any -f "host api.openai.com"

Check disk usage for logs and cache
du -sh ~/.config/perspt/
```

### Resource Optimization

```
{
 "resource_limits": {
 "max_history_size": 50, // Limit conversation history
 "cache_size_mb": 100, // Limit cache size
 "log_rotation_size": "10MB", // Rotate logs
 "cleanup_interval": "24h" // Regular cleanup
 }
}
```

## Network Performance

### Connection Optimization

```
{
 "network": {
 "keep_alive": true, // Reuse connections
 "connection_pool_size": 5, // Pool connections
 "dns_cache": true, // Cache DNS lookups
 "compression": true // Enable compression
 }
}
```

### Regional Configuration

```
{
 "provider_endpoints": {
 "openai": "https://api.openai.com", // US
 "anthropic": "https://api.anthropic.com", // US
 "google": "https://generativelanguage.googleapis.com" // Global
 }
}
```



## Advanced Recovery Procedures

### Complete System Reset

#### Full Configuration Reset

```
Backup current configuration
cp -r ~/.config/perspt ~/.config/perspt.backup.$(date +%Y%m%d)

Remove all Perspt data
rm -rf ~/.config/perspt/
rm -rf ~/.local/share/perspt/
rm -rf ~/.cache/perspt/

Clear temporary files
rm -rf /tmp/perspt*

Recreate default configuration
perspt --create-default-config
```

#### Selective Reset Options

```
Reset only configuration
rm ~/.config/perspt/config.json
perspt --setup

Clear only cache
rm -rf ~/.config/perspt/cache/

Clear only conversation history
rm -rf ~/.config/perspt/history/

Reset only logs
rm ~/.config/perspt/*.log
```

## Emergency Fallback Procedures

### Provider Fallback Chain

```
{
 "fallback_chain": [
 {
 "provider_type": "openai",
 "model": "gpt-4o-mini",
 "on_failure": "next"
 },
 {
 "provider_type": "anthropic",
 "model": "claude-3-5-haiku-20241022",
 "on_failure": "next"
 },
 {
 "provider_type": "ollama",
 "model": "llama3.2:8b",

```

(continues on next page)

(continued from previous page)

```
 "on_failure": "fail"
 }
}
}
```

## Manual Override Mode

```
Force specific provider regardless of config
perspt --force-provider openai --force-model gpt-3.5-turbo

Use minimal configuration
perspt --no-config --api-key sk-... --provider-type openai

Debug mode with maximum verbosity
perspt --debug --verbose --log-level trace
```

## Data Recovery

### Conversation History Recovery

```
Check for backup files
ls ~/.config/perspt/history/*.backup

Restore from backup
cp ~/.config/perspt/history/conversation.backup \
 ~/.config/perspt/history/conversation.json

Export conversations before reset
perspt --export-history ~/perspt-backup.json
```

### Configuration Recovery

```
Restore from automatic backup
cp ~/.config/perspt/config.json.backup ~/.config/perspt/config.json

Recreate from environment variables
perspt --config-from-env

Interactive configuration rebuild
perspt --reconfigure
```

## Version Migration Issues

### Upgrading from allms to genai

```
Backup old configuration
cp ~/.config/perspt/config.json ~/.config/perspt/config.allms.backup

Run migration script
perspt --migrate-config
```

(continues on next page)

(continued from previous page)

```
Manual migration if needed
perspt --validate-config --fix-issues
```

## Downgrade Procedures

```
Install specific version
cargo install perspt --version 0.2.0

Use version-specific configuration
cp ~/.config/perspt/config.v0.2.0.json ~/.config/perspt/config.json
```

## Emergency Contact and Support

### Critical Issue Escalation

For production-critical issues:

1. **Immediate Workarounds:** - Switch to backup providers - Use local models (Ollama) for offline capability - Enable debug logging for detailed diagnosis
2. **Community Support Channels:** - GitHub Issues: <https://github.com/eonseed/perspt/issues> - Discord Community: [Link to Discord] - Reddit: r/perspt
3. **Enterprise Support:** - Priority ticket system - Direct developer contact - Custom configuration assistance

### Issue Documentation Template

When reporting issues, include this information:

```
Environment Information:
- OS: [macOS 14.1 / Ubuntu 22.04 / Windows 11]
- Perspt Version: [perspt --version]
- Installation Method: [cargo / brew / binary]

Configuration:
- Provider: [openai / anthropic / google / etc.]
- Model: [gpt-4o-mini / claude-3-5-sonnet / etc.]
- Config file: [attach sanitized config.json]

Error Details:
- Full error message: [exact text]
- Error code: [if available]
- Stack trace: [if available]

Reproduction Steps:
1. [Step 1]
2. [Step 2]
3. [Error occurs]

Expected vs Actual Behavior:
- Expected: [what should happen]
- Actual: [what actually happens]

Additional Context:
```

(continues on next page)

(continued from previous page)

- Network environment: [corporate / home / proxy]
- Recent changes: [configuration / system updates]
- Workarounds attempted: [list what you've tried]

## Recovery Verification

After resolving issues, verify system health:

```
Test basic functionality
perspt --provider-type openai --model gpt-3.5-turbo --test-connection

Verify configuration
perspt --validate-config

Test streaming
echo "Hello" | perspt --provider-type openai --model gpt-4o-mini --stream

Check all providers
for provider in openai anthropic google groq; do
 echo "Testing $provider..."
 perspt --provider-type $provider --list-models
done
```

## Related Documentation


For additional help:

- *AI Providers* - Provider-specific configuration and features
- *Advanced Features* - Advanced usage patterns and optimization
- *Configuration Guide* - Complete configuration reference
- *Developer Guide* - Development and API documentation
- *API Reference* - API reference and integration guides

## 1.7 Concepts

Understanding the foundations of Perspt.

- **SRBN Architecture** The Stabilized Recursive Barrier Network engine for autonomous coding.

*SRBN Architecture*  **Workspace Crates** The 6-crate modular architecture.

*Workspace Crates*  **PSP Process** Perspt Specification Proposals for design and development.

*PSP: Perspt Specification Proposals*

### 1.7.1 SRBN Architecture

The **Stabilized Recursive Barrier Network (SRBN)** is Perspt's core innovation for autonomous coding with mathematically guaranteed stability.

## Overview

SRBN ensures that AI-generated code converges to a stable state before being committed, using concepts from control theory (Lyapunov stability) and software verification (LSP, tests).

## The Control Loop

The SRBN control loop executes 5 phases for each task:

| # | Phase                 | Description                                                                    |
|---|-----------------------|--------------------------------------------------------------------------------|
| 1 | <b>Sheafification</b> | Architect model decomposes task into a JSON TaskPlan with dependency graph     |
| 2 | <b>Speculation</b>    | Actuator model generates code for each node with tool calls (write_file, etc.) |
| 3 | <b>Verification</b>   | Compute Lyapunov Energy $V(x)$ from LSP diagnostics, structure, and tests      |
| 4 | <b>Convergence</b>    | If $V(x) > \varepsilon$ , retry with error feedback; otherwise proceed         |
| 5 | <b>Commit</b>         | Record changes in Merkle Ledger with cryptographic integrity                   |

## Lyapunov Energy

The stability of generated code is measured using a Lyapunov energy function:

### Energy Formula

$$V(x) = \alpha \cdot V_{\text{syn}} + \beta \cdot V_{\text{str}} + \gamma \cdot V_{\text{log}}$$

Default weights:  $\alpha = 1.0$ ,  $\beta = 0.5$ ,  $\gamma = 2.0$

## Components

| Component    | Source              | Description                                                             |
|--------------|---------------------|-------------------------------------------------------------------------|
| <b>V_syn</b> | LSP Diagnostics     | Count of errors and warnings from <code>ty</code> (Python type checker) |
| <b>V_str</b> | Structural Analysis | Code complexity, dead code, pattern violations                          |
| <b>V_log</b> | Test Failures       | Weighted sum of pytest failures (critical tests have higher weight)     |

## Convergence Criterion

The system is considered stable when:

$$V(x) \leq \varepsilon$$

Default:  $\varepsilon = 0.1$

## Model Tiers

SRBN uses multiple specialized models:

| Tier              | Purpose                            | Recommended Model          |
|-------------------|------------------------------------|----------------------------|
| <b>Architect</b>  | Deep reasoning, task decomposition | GPT-5.2, Claude Opus 4.5   |
| <b>Actuator</b>   | Code generation, tool calls        | Claude Opus 4.5, GPT-5.2   |
| <b>Verifier</b>   | Stability analysis                 | Gemini 3 Pro               |
| <b>Speculator</b> | Fast lookahead, branch prediction  | Gemini 3 Flash, Groq Llama |

Configure model tiers via CLI:

```
perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model claude-opus-4.5 \
 --verifier-model gemini-3-pro \
 --speculator-model gemini-3-flash \
 "Build a REST API"
```

### Retry Policy

SRBN implements bounded retries per PSP-0004:

| Error Type               | Max Retries | Escalation                    |
|--------------------------|-------------|-------------------------------|
| Compilation errors (LSP) | 3           | Escalate to user with context |
| Tool failures (file ops) | 5           | Escalate with error logs      |
| Review rejections        | 3           | Escalate with diff summary    |

### TaskPlan Structure

The Architect generates a JSON TaskPlan:

```
{
 "nodes": [
 {
 "id": 1,
 "description": "Create Calculator class",
 "type": "create",
 "files": ["calculator.py"],
 "dependencies": []
 },
 {
 "id": 2,
 "description": "Add arithmetic methods",
 "type": "modify",
 "files": ["calculator.py"],
 "dependencies": [1]
 },
 {
 "id": 3,
 "description": "Write unit tests",
 "type": "create",
```

(continues on next page)

(continued from previous page)

```

 "files": ["test_calculator.py"],
 "dependencies": [2]
 }
]
}

```

### Merkle Ledger

All changes are recorded in a Merkle tree for:

- **Integrity** — Cryptographic verification of change history
- **Rollback** — Revert to any previous state
- **Audit** — Complete trail of AI-generated changes

```

View recent commits
perspt ledger --recent

Rollback to commit
perspt ledger --rollback abc123

Statistics
perspt ledger --stats

```

### See Also

- *PSP: Perspt Specification Proposals* - The PSP-0004 specification
- *perspt-agent API* - API documentation
- *Agent Mode Tutorial* - Tutorial walkthrough

## 1.7.2 Workspace Crates

Perspt is organized as a **6-crate Cargo workspace** for modularity and maintainability.

### Crate Overview

#### Crate Details

##### `perspt-cli`

**Purpose:** Command-line interface entry point

**Location:** `crates/perspt-cli/`

**Key Components:**

- `main.rs` — CLI argument parsing with clap
- `commands/` — Subcommand implementations

**Subcommands:**

| Command | Function                   |
|---------|----------------------------|
| chat    | Launch interactive TUI     |
| agent   | Run SRBN orchestrator      |
| init    | Initialize project config  |
| config  | Manage configuration       |
| ledger  | Query Merkle ledger        |
| status  | Show agent status          |
| abort   | Cancel current session     |
| resume  | Resume interrupted session |

### **perspt-core**

**Purpose:** Core abstractions for LLM interaction

**Location:** crates/perspt-core/

**Key Components:**

- `config.rs` — Simple Config struct
- `llm_provider.rs` — Thread-safe GenAIProvider
- `memory.rs` — Conversation memory

**Thread Safety:** GenAIProvider uses `Arc<RwLock<SharedState>>` for concurrent access.

### **perspt-agent**

**Purpose:** SRBN autonomous coding engine

**Location:** crates/perspt-agent/

**Key Components:**

| Module                            | Size | Description                               |
|-----------------------------------|------|-------------------------------------------|
| <code>orchestrator.rs</code>      | 34KB | SRBN control loop, model tiers            |
| <code>lsp.rs</code>               | 28KB | LSP client for Python ( <code>ty</code> ) |
| <code>tools.rs</code>             | 12KB | Agent tools (read, write, search, shell)  |
| <code>types.rs</code>             | 24KB | TaskPlan, Node, Energy types              |
| <code>ledger.rs</code>            | 6KB  | Merkle change tracking                    |
| <code>test_runner.rs</code>       | 15KB | pytest integration                        |
| <code>context_retriever.rs</code> | 10KB | Code context extraction                   |

### **perspt-tui**

**Purpose:** Terminal UI components

**Location:** crates/perspt-tui/

**Key Components:**

- `agent_app.rs` — Main agent TUI
- `dashboard.rs` — Status metrics
- `diff_viewer.rs` — File diff display
- `review_modal.rs` — Change approval
- `task_tree.rs` — Task hierarchy



### [perspt-policy](#)

**Purpose:** Security policy enforcement

**Location:** crates/perspt-policy/

**Key Components:**

- `engine.rs` — Starlark policy evaluator
- `sanitize.rs` — Command sanitization

### [perspt-sandbox](#)

**Purpose:** Process isolation

**Location:** crates/perspt-sandbox/

**Key Component:** `command.rs` — Sandboxed command execution with resource limits

### Building Individual Crates

```
Build specific crate
cargo build -p perspt-cli
cargo build -p perspt-agent

Run tests for crate
cargo test -p perspt-core

Generate docs for crate
cargo doc -p perspt-agent --open
```

### Adding a New Crate

1. Create directory: `crates/perspt-newcrate/`
2. Add `Cargo.toml` with package metadata
3. Register in root `Cargo.toml`:

```
[workspace]
members = [
 "crates/perspt-core",
 "crates/perspt-newcrate", # Add here
 ...
]
```

4. Add dependencies to consuming crates

### See Also

- [Architecture](#) - Architecture overview
- [Extending Perspt](#) - Extension guide
- [API Reference](#) - Per-crate API reference

## 1.7.3 PSP: Perspt Specification Proposals

PSP (Perspt Specification Proposals) is the process by which Perspt designs and implements new features in a structured, reviewable manner.

## Overview

PSPs are design documents that describe proposed features, architectural changes, or process improvements for Perspt. They are inspired by PEPs (Python Enhancement Proposals) and RFCs (Request for Comments).

## PSP Format

Each PSP follows a standard format in reStructuredText:

```
:PSP: 000N
>Title: Feature Name
:Author: Name <email@example.com>
>Status: Draft | Under Review | Accepted | Implemented | Rejected
:Created: YYYY-MM-DD
:Updated: YYYY-MM-DD
```

### Abstract

One-paragraph summary.

### Motivation

Why is this needed?

### Specification

Technical details.

### Implementation

How will it be implemented?

### Rationale

Why this approach?

### Reference Implementation

Links to code.

## Current PSPs

| PSP      | Title                       | Status      | Description                          |
|----------|-----------------------------|-------------|--------------------------------------|
| PSP-0001 | PSP Process                 | Implemented | This document                        |
| PSP-0002 | Multi-Provider Architecture | Implemented | GenAI-based unified provider         |
| PSP-0003 | Configuration System        | Implemented | JSON config with env vars            |
| PSP-0004 | SRBN Agent Mode             | Implemented | Stabilized Recursive Barrier Network |

### PSP-0004: SRBN Agent Mode

The most significant PSP, introducing autonomous coding capabilities.

#### Key Components:

1. **Orchestrator** — SRBN control loop
2. **Lyapunov Energy** — Stability measurement ( $V_{\text{syn}}$ ,  $V_{\text{str}}$ ,  $V_{\text{log}}$ )
3. **Model Tiers** — Architect, Actuator, Verifier, Speculator
4. **Retry Policy** — Bounded retries with escalation
5. **Merkle Ledger** — Change tracking with rollback

#### State Machine:

#### Creating a PSP

1. **Assign Number**: Get the next PSP number from the maintainers
2. **Create File**: docs/psps/source/psp-00000N.rst
3. **Write Draft**: Follow the template format
4. **Submit PR**: Open a pull request with the PSP
5. **Review**: Gather feedback and iterate
6. **Accept/Reject**: Maintainers decide on the proposal

#### PSP Repository

All PSPs are stored in the docs/psps/ directory:


```
docs/psps/
├── source/
│ ├── psp-000001.rst # PSP Process
│ ├── psp-000002.rst # Multi-Provider
│ ├── psp-000003.rst # Configuration
│ └── psp-000004.rst # SRBN Agent Mode
└── Makefile
```


#### See Also

- [PSP Repository](#)
- [SRBN Architecture](#) - Technical details of SRBN
- [Contributing](#) - How to contribute


## 1.8 How-To Guides

Task-oriented guides for common operations.

 **Configuration** Set up providers, models, and preferences.

*Configuration*  **Providers** Connect to OpenAI, Anthropic, Google, and more.

*Providers*  **Agent Options** Configure SRBN agent parameters.

*Agent Options*  **Security Rules** Set up Starlark policy rules.

*Security Rules*

## 1.8.1 Configuration

How to configure Perspt for your workflow.

### Configuration Sources

Perspt loads configuration from (highest priority first):

1. **CLI Arguments** — `perspt --model gpt-5.2`
2. **Environment Variables** — `export OPENAI_API_KEY=...`
3. **Config File** — `~/.perspt/config.toml`
4. **Defaults** — Built-in fallbacks

### Config File Location

```
~/.perspt/config.toml
```

Create it:

```
mkdir -p ~/.perspt
perspt config --edit
```

### Config File Format

```
~/.perspt/config.toml

[default]
provider = "openai"
model = "gpt-5.2"

[providers.openai]
api_key = "sk-..."

[providers.anthropic]
api_key = "sk-ant-..."

[agent]
architect_model = "gpt-5.2"
actuator_model = "claude-opus-4.5"
verifier_model = "gemini-3-pro"
energy_weights = [1.0, 0.5, 2.0]
stability_threshold = 0.1
max_retries_compile = 3
max_retries_tool = 5
```

## Environment Variables

| Variable          | Description           |
|-------------------|-----------------------|
| OPENAI_API_KEY    | OpenAI API key        |
| ANTHROPIC_API_KEY | Anthropic API key     |
| GEMINI_API_KEY    | Google Gemini API key |
| GROQ_API_KEY      | Groq API key          |
| COHERE_API_KEY    | Cohere API key        |
| XAI_API_KEY       | XAI (Grok) API key    |
| DEEPSEEK_API_KEY  | DeepSeek API key      |

## CLI Configuration Commands

```
Show current configuration
perspt config --show

Set a value
perspt config --set default.model=gpt-5.2

Edit in $EDITOR
perspt config --edit
```

## Project Configuration

Initialize project-specific config:

```
cd my-project
perspt init --memory --rules
```

This creates:

```
my-project/
├── PERSPT.md # Project memory/context
└── .perspt/
 ├── config.toml # Project config
 └── rules.star # Policy rules
```

## PERSPT.md

Project memory file that provides context to the agent:

```
My Project

Overview
This is a Python web application using FastAPI.

Architecture
- `api/` - REST endpoints
- `core/` - Business logic
- `tests/` - pytest suite
```

(continues on next page)

(continued from previous page)

**## Conventions**

- Use type hints everywhere
- 100% test coverage required

**Per-Session Configuration**

Override for a single session:

```
perspt chat --model claude-opus-4.5

perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model claude-opus-4.5 \
 "Create module"
```

**See Also**

- *Providers* - Provider-specific setup
- *Agent Options* - Agent configuration
- *Security Rules* - Policy rules

## 1.8.2 Providers

How to set up each LLM provider with Perspt.

**OpenAI**

**Models:** GPT-5.2, o3-mini, o1-preview, GPT-4

```
Set API key
export OPENAI_API_KEY="sk-..."

Use with Perspt
perspt chat --model gpt-5.2
```

**Get API Key:** [platform.openai.com](https://platform.openai.com)

**Anthropic**

**Models:** Claude Opus 4.5, Claude 3.5 Sonnet

```
export ANTHROPIC_API_KEY="sk-ant-..."
perspt chat --model claude-opus-4.5
```

**Get API Key:** [console.anthropic.com](https://console.anthropic.com)

**Google Gemini**

**Models:** Gemini 3 Flash, Gemini 3 Pro

```
export GEMINI_API_KEY="..."
perspt chat --model gemini-3-flash
```

**Get API Key:** [aistudio.google.com](https://aistudio.google.com)

### Groq

**Models:** Llama 3.x (ultra-fast inference)

```
export GROQ_API_KEY="..."
perspt chat --model llama-3.3-70b
```

**Get API Key:** [console.groq.com](https://console.groq.com)

**Best for:** Fast prototyping, testing

### Cohere

**Models:** Command R, Command R+

```
export COHERE_API_KEY="..."
perspt chat --model command-r-plus
```

**Get API Key:** [dashboard.cohere.com](https://dashboard.cohere.com)

### XAI (Grok)

**Models:** Grok

```
export XAI_API_KEY="..."
perspt chat --model grok-2
```

**Get API Key:** [console.x.ai](https://console.x.ai)

### DeepSeek

**Models:** DeepSeek Coder, DeepSeek Chat

```
export DEEPSEEK_API_KEY="..."
perspt chat --model deepseek-coder
```

**Get API Key:** [platform.deepseek.com](https://platform.deepseek.com)

### Ollama (Local)

**Models:** Llama 3.2, CodeLlama, DeepSeek Coder (local)

```
No API key needed
ollama serve
ollama pull llama3.2
perspt chat --model llama3.2
```

**Setup:** See *Local Models with Ollama*

## Provider Comparison

| Provider  | Speed      | Best For                 | Cost   |
|-----------|------------|--------------------------|--------|
| OpenAI    | Medium     | Reasoning, complex tasks | \$\$\$ |
| Anthropic | Medium     | Code generation, safety  | \$\$\$ |
| Google    | Fast       | Long context, multimodal | \$\$   |
| Groq      | Ultra-fast | Prototyping, testing     | \$     |
| Ollama    | Variable   | Privacy, offline use     | Free   |

## Agent Mode Recommendations

For optimal SRBN performance:

```
perspt agent \
 --architect-model gpt-5.2 \ # Deep reasoning
 --actuator-model claude-opus-4.5 \ # Strong coding
 --verifier-model gemini-3-pro \ # Fast analysis
 --speculator-model gemini-3-flash \ # Ultra-fast
 "Your task"
```

## See Also

- *Configuration* - Config file setup
- *Local Models with Ollama* - Ollama guide

## 1.8.3 Agent Options

Complete reference for SRBN agent configuration.

### Basic Usage

```
perspt agent [OPTIONS] <TASK>
```

### Required Arguments

|        |                                       |
|--------|---------------------------------------|
| <TASK> | Task description or path to task file |
|--------|---------------------------------------|

### Model Selection

| Option                     | Description                                   |
|----------------------------|-----------------------------------------------|
| --model <MODEL>            | Override ALL model tiers                      |
| --architect-model <MODEL>  | Model for task decomposition (deep reasoning) |
| --actuator-model <MODEL>   | Model for code generation                     |
| --verifier-model <MODEL>   | Model for stability checking                  |
| --speculator-model <MODEL> | Model for fast lookahead                      |

**Example:**



```
perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model claude-opus-4.5 \
 "Build REST API"
```

## Execution Control

| Option               | Description                                   |
|----------------------|-----------------------------------------------|
| -w, --workdir <DIR>  | Working directory (default: current)          |
| -y, --yes            | Auto-approve all actions                      |
| --auto-approve-safe  | Auto-approve read-only operations only        |
| -k, --complexity <K> | Max tasks before approval prompt (default: 5) |
| --mode <MODE>        | Execution mode: cautious, balanced, yolo      |

### Modes:

|          |                                        |
|----------|----------------------------------------|
| cautious | Prompt for every change                |
| balanced | Prompt when complexity > K (default)   |
| yolo     | Auto-approve everything (☐☐ dangerous) |

## SRBN Parameters

| Option                                       | Description                               |
|----------------------------------------------|-------------------------------------------|
| --energy-weights < $\alpha, \beta, \gamma$ > | Lyapunov weights (default: 1.0, 0.5, 2.0) |
| --stability-threshold < $\epsilon$ >         | Convergence threshold (default: 0.1)      |

**Energy Formula:**  $V(x) = \alpha \cdot V_{\text{syn}} + \beta \cdot V_{\text{str}} + \gamma \cdot V_{\text{log}}$

### Tuning Examples:

```
Prioritize tests (raise γ)
perspt agent --energy-weights "1.0,0.5,3.0" "Add tests"

Prioritize type safety (raise α)
perspt agent --energy-weights "2.0,0.5,1.0" "Add type hints"

More lenient (raise ϵ)
perspt agent --stability-threshold 0.5 "Quick prototype"
```

## Limits

| Option           | Description                             |
|------------------|-----------------------------------------|
| --max-cost <USD> | Maximum cost in dollars (0 = unlimited) |
| --max-steps <N>  | Maximum iterations (0 = unlimited)      |

### Example:

```
perspt agent --max-cost 5.0 --max-steps 20 "Large refactor"
```

### Session Management

```
Check current status
perspt status

Cancel current session
perspt abort
perspt abort --force # No confirmation

Resume interrupted session
perspt resume
perspt resume <session_id>
```

### Ledger Operations

```
View recent changes
perspt ledger --recent

Rollback to commit
perspt ledger --rollback <hash>

Statistics
perspt ledger --stats
```

### Full Examples

#### Conservative approach:

```
perspt agent \
 --mode cautious \
 -k 1 \
 --max-cost 1.0 \
 --max-steps 10 \
 -w ./project \
 "Add input validation"
```

#### Fast prototyping:

```
perspt agent -y \
 --model gemini-3-flash \
 --stability-threshold 0.5 \
 "Create boilerplate"
```

#### Production-grade:

```
perspt agent \
 --architect-model gpt-5.2 \
 --actuator-model claude-opus-4.5 \
 --verifier-model gemini-3-pro \
 --energy-weights "2.0,1.0,3.0" \
```

(continues on next page)

(continued from previous page)

```
--stability-threshold 0.05 \
--max-cost 10.0 \
-w ./project \
"Implement authentication system"
```

### See Also

- *SRBN Architecture* - SRBN details
- *Agent Mode Tutorial* - Tutorial
- *Configuration* - Config file

## 1.8.4 Security Rules

How to configure Starlark policy rules for agent operations.

### Overview

The policy engine evaluates commands before execution using Starlark rules. This prevents dangerous operations and provides fine-grained control.

### Initialize Rules

```
perspt init --rules
```

This creates `.perspt/rules.star`:

```
my-project/
├── .perspt/
│ └── rules.star
```

### Rule Syntax

Rules use three functions:

```
allow("pattern") # Always allow
prompt("pattern", reason="...") # Ask user
deny("pattern", reason="...") # Always deny
```

**Patterns** support glob syntax:

- `*` — Any characters
- `?` — Single character
- `[abc]` — Character set

### Default Rules

```
.perspt/rules.star

Safe read operations
allow("cat *")
allow("head *")
allow("tail *")
```

(continues on next page)

(continued from previous page)

```
allow("ls *")
allow("find *")
allow("grep *")

Prompt for modifications
prompt("rm *", reason="File deletion")
prompt("mv *", reason="File move/rename")
prompt("cp *", reason="File copy")
prompt("chmod *", reason="Permission change")

Deny dangerous operations
deny("rm -rf /", reason="Root deletion")
deny("rm -rf ~", reason="Home deletion")
deny("rm -rf /*", reason="Wildcard root deletion")
deny("chmod 777 *", reason="Insecure permissions")
deny("curl * | bash", reason="Remote code execution")
deny("wget * | bash", reason="Remote code execution")
```

## Custom Rules

For a Node.js project:

```
.perspt/rules.star

Allow package management
allow("npm install")
allow("npm test")
allow("npm run *")

Prompt for global operations
prompt("npm install -g *", reason="Global install")
prompt("npm uninstall *", reason="Package removal")

Deny destructive
deny("rm -rf node_modules", reason="Use npm prune instead")
```

For a Python project:

```
.perspt/rules.star

Allow common operations
allow("python *")
allow("pip install *")
allow("pytest *")
allow("uv *")

Prompt for system changes
prompt("pip uninstall *", reason="Package removal")
prompt("pip install --user *", reason="User install")

Deny dangerous
deny("pip install --break-system-packages *")
```

## Testing Rules

Test your rules before deployment:

```
Simulate a command
perspt policy test "rm -rf /"
Output: DENIED - Root deletion

perspt policy test "cat README.md"
Output: ALLOWED
```

## Project-Specific Overrides

Rules are inherited hierarchically:

1. Global: ~/.perspt/rules.star
2. Project: .perspt/rules.star

Project rules override global rules.

## Bypass for Trusted Tasks

Use `-y` or `--mode yo!o` to skip policy checks (☐☐ dangerous):

```
perspt agent -y "Install dependencies"
```

## See Also

- *perspt-policy API* - Policy engine API
- *Agent Options* - Agent configuration

# 1.9 Configuration Guide

Perspt offers flexible configuration options to customize your AI chat experience. This guide covers all configuration methods, from zero-config automatic provider detection to advanced JSON configurations.

## 1.9.1 Automatic Provider Detection (Zero-Config)

**NEW!** Perspt now features intelligent automatic provider detection that makes getting started as simple as setting an environment variable.

### How It Works

When you launch Perspt without specifying a provider, it automatically scans your environment variables for supported provider API keys and selects the first one found based on this priority order:

1. **OpenAI** (OPENAI\_API\_KEY) - Default model: *gpt-4o-mini*
2. **Anthropic** (ANTHROPIC\_API\_KEY) - Default model: *claude-3-5-sonnet-20241022*
3. **Google Gemini** (GEMINI\_API\_KEY) - Default model: *gemini-1.5-flash*
4. **Groq** (GROQ\_API\_KEY) - Default model: *llama-3.1-70b-versatile*
5. **Cohere** (COHERE\_API\_KEY) - Default model: *command-r-plus*
6. **XAI** (XAI\_API\_KEY) - Default model: *grok-beta*
7. **DeepSeek** (DEEPSEEK\_API\_KEY) - Default model: *deepseek-chat*
8. **Ollama** (auto-detected if running) - Default model: *llama3.2*

### Quick Examples

```
Option 1: OpenAI (highest priority)
export OPENAI_API_KEY="sk-your-key"
perspt # Auto-detects OpenAI, uses gpt-4o-mini

Option 2: Multiple providers - OpenAI takes priority
export OPENAI_API_KEY="sk-your-openai-key"
export ANTHROPIC_API_KEY="sk-ant-your-anthropic-key"
perspt # Uses OpenAI (higher priority)

Option 3: Force a specific provider
perspt --provider anthropic # Uses Anthropic even if OpenAI key exists

Option 4: Ollama (no API key needed)
Just ensure Ollama is running: ollama serve
perspt # Auto-detects Ollama if no other keys found
```

### What Happens When No Providers Are Found

If no API keys are detected, Perspt displays helpful setup instructions:

```
❑ No LLM provider configured!
```

To get started, either:

1. Set an environment variable for a supported provider:
  - OPENAI\_API\_KEY=sk-your-key
  - ANTHROPIC\_API\_KEY=sk-ant-your-key
  - GEMINI\_API\_KEY=your-key

```
... (shows all supported providers)
```
2. Use command line arguments:

```
perspt --provider openai --api-key sk-your-key
```

## 1.9.2 Manual Configuration Methods

For more control or advanced setups, Perspt supports traditional configuration methods with the following priority order (highest to lowest):

1. **Command-line arguments** (highest priority)
2. **Configuration file** (config.json)
3. **Environment variables**
4. **Automatic provider detection**
5. **Default values** (lowest priority)

This means command-line arguments will override config file settings, which override environment variables, and so on.

## 1.9.3 Environment Variables

Environment variables are the simplest way to configure Perspt and enable automatic provider detection.

### API Keys (Auto-Detection Enabled)

Setting any of these environment variables enables automatic provider detection:

```
OpenAI (Priority 1 - will be auto-selected first)
export OPENAI_API_KEY="sk-your-openai-api-key-here"

Anthropic (Priority 2)
export ANTHROPIC_API_KEY="your-anthropic-api-key-here"

Google Gemini (Priority 3)
export GEMINI_API_KEY="your-google-api-key-here"

Groq (Priority 4)
export GROQ_API_KEY="your-groq-api-key-here"

Cohere (Priority 5)
export COHERE_API_KEY="your-cohere-api-key-here"

XAI (Priority 6)
export XAI_API_KEY="your-xai-api-key-here"

DeepSeek (Priority 7)
export DEEPSEEK_API_KEY="your-deepseek-api-key-here"

Ollama (Priority 8 - no API key needed, auto-detected if service is running)
Just run: ollama serve
```

#### Note

**Automatic Detection:** Simply set any of these environment variables and run perspt with no arguments. Perspt will automatically detect and use the highest priority provider available.

### Legacy Provider Settings (Manual Override)

These variables override automatic detection and force manual configuration:

```
Default provider
export PERSPT_PROVIDER="openai"

Default model
export PERSPT_MODEL="gpt-4o-mini"

Custom API base URL
export PERSPT_API_BASE="https://api.openai.com/v1"
```

## 1.9.4 Configuration File

For persistent settings, create a config.json file:

### Basic Configuration

```
{
 "api_key": "your-api-key-here",
 "default_model": "gpt-4o-mini",
 "default_provider": "openai",
 "provider_type": "openai"
}
```

### Complete Configuration

```
{
 "api_key": "sk-your-openai-api-key",
 "default_model": "gpt-4o-mini",
 "default_provider": "openai",
 "provider_type": "openai",
 "providers": {
 "openai": "https://api.openai.com/v1",
 "anthropic": "https://api.anthropic.com",
 "google": "https://generativelanguage.googleapis.com/v1beta",
 "azure": "https://your-resource.openai.azure.com/"
 },
 "ui": {
 "theme": "dark",
 "show_timestamps": true,
 "markdown_rendering": true,
 "auto_scroll": true
 },
 "behavior": {
 "stream_responses": true,
 "input_queuing": true,
 "auto_save_history": false,
 "max_history_length": 1000
 },
 "advanced": {
 "request_timeout": 30,
 "retry_attempts": 3,
 "retry_delay": 1.0,
 "concurrent_requests": 1
 }
}
```

### Configuration File Locations

Perspt searches for configuration files in this order:

1. **Specified path:** `perspt --config /path/to/config.json`
2. **Current directory:** `./config.json`
3. **User config directory:** - Linux: `~/.config/perspt/config.json` - macOS: `~/Library/Application Support/perspt/config.json` - Windows: `%APPDATA%/perspt/config.json`



## 1.9.5 Provider Configuration

### OpenAI

Environment Variables

Config File

Command Line

```
export OPENAI_API_KEY="sk-your-key-here"
export PERSPT_PROVIDER="openai"
export PERSPT_MODEL="gpt-4o-mini"
```

```
{
 "api_key": "sk-your-key-here",
 "provider_type": "openai",
 "default_model": "gpt-4o-mini",
 "providers": {
 "openai": "https://api.openai.com/v1"
 }
}
```

```
perspt --provider-type openai \
 --model-name gpt-4o-mini \
 --api-key "sk-your-key-here"
```

**Available Models:** - gpt-4.1 - Latest and most advanced GPT model - gpt-4o - Latest GPT-4 Omni model - gpt-4o-mini - Faster, cost-effective GPT-4 Omni - o1-preview - Advanced reasoning model - o1-mini - Efficient reasoning model - o3-mini - Next-generation reasoning model - gpt-4-turbo - Latest GPT-4 Turbo - gpt-4 - Standard GPT-4

### Anthropic

Environment Variables

Config File

Command Line

```
export ANTHROPIC_API_KEY="your-key-here"
export PERSPT_PROVIDER="anthropic"
export PERSPT_MODEL="claude-3-sonnet-20240229"
```

```
{
 "api_key": "your-key-here",
 "provider_type": "anthropic",
 "default_model": "claude-3-sonnet-20240229",
 "providers": {
 "anthropic": "https://api.anthropic.com"
 }
}
```

```
perspt --provider-type anthropic \
 --model-name claude-3-sonnet-20240229 \
 --api-key "your-key-here"
```

**Available Models:** - claude-3-opus-20240229 - Most capable Claude model - claude-3-sonnet-20240229 - Balanced performance and speed - claude-3-haiku-20240307 - Fastest Claude model

Google (Gemini)

Environment Variables

Config File

Command Line

```
export GOOGLE_API_KEY="your-key-here"
export PERSPT_PROVIDER="google"
export PERSPT_MODEL="gemini-pro"
```

```
{
 "api_key": "your-key-here",
 "provider_type": "google",
 "default_model": "gemini-pro",
 "providers": {
 "google": "https://generativelanguage.googleapis.com/v1beta"
 }
}
```

```
perspt --provider-type google \
 --model-name gemini-pro \
 --api-key "your-key-here"
```

**Available Models:** - gemini-pro - Google’s most capable model - gemini-pro-vision - Multimodal capabilities

1.9.6 Command-Line Options

Perspt supports extensive command-line configuration:

Basic Options

```
perspt [OPTIONS]
```

| Option                 | Description                                                                  |
|------------------------|------------------------------------------------------------------------------|
| --config <PATH>        | Path to configuration file                                                   |
| --provider-type <TYPE> | AI provider (openai, anthropic, google, groq, cohere, xai, deepseek, ollama) |
| --model-name <MODEL>   | Specific model to use                                                        |
| --api-key <KEY>        | API key for authentication                                                   |
| --list-models          | List available models for provider                                           |
| --help                 | Show help information                                                        |
| --version              | Show version information                                                     |

Advanced Options

```
Custom API endpoint
perspt --api-base "https://your-custom-endpoint.com/v1"
```

(continues on next page)

(continued from previous page)

```
Increase request timeout
perspt --timeout 60

Disable streaming responses
perspt --no-stream

Set maximum retries
perspt --max-retries 5

Custom user agent
perspt --user-agent "MyApp/1.0"
```

## Examples

```
Use specific OpenAI model
perspt --provider-type openai --model-name gpt-4

Use Anthropic with custom timeout
perspt --provider-type anthropic \
 --model-name claude-3-sonnet-20240229 \
 --timeout 45

Use custom configuration file
perspt --config ~/.perspt/work-config.json

List available models
perspt --provider-type openai --list-models
```

## 1.9.7 UI Customization

### Interface Settings

Configure the terminal interface appearance:

```
{
 "ui": {
 "theme": "dark",
 "show_timestamps": true,
 "timestamp_format": "%H:%M",
 "markdown_rendering": true,
 "syntax_highlighting": true,
 "auto_scroll": true,
 "scroll_buffer": 1000,
 "word_wrap": true,
 "show_token_count": false
 }
}
```

## Color Themes

Customize colors for different message types:

```
{
 "ui": {
 "colors": {
 "user_message": "#60a5fa",
 "assistant_message": "#10b981",
 "error_message": "#ef4444",
 "warning_message": "#f59e0b",
 "info_message": "#8b5cf6",
 "timestamp": "#6b7280",
 "border": "#374151",
 "background": "#111827"
 }
 }
}
```

## 1.9.8 Behavior Settings

### Streaming and Responses

```
{
 "behavior": {
 "stream_responses": true,
 "input_queuing": true,
 "auto_retry_on_error": true,
 "show_thinking_indicator": true,
 "preserve_context": true
 }
}
```

### History Management

```
{
 "behavior": {
 "auto_save_history": true,
 "history_file": "~/.perspt/chat_history.json",
 "max_history_length": 1000,
 "history_compression": true,
 "clear_history_on_exit": false
 }
}
```

## 1.9.9 Advanced Configuration

### Network Settings

```
{
 "advanced": {
 "request_timeout": 30,
```

(continues on next page)

(continued from previous page)

```

 "connect_timeout": 10,
 "retry_attempts": 3,
 "retry_delay": 1.0,
 "retry_exponential_backoff": true,
 "max_concurrent_requests": 1,
 "user_agent": "Perspt/0.4.0",
 "proxy": {
 "http": "http://proxy:8080",
 "https": "https://proxy:8080"
 }
 }
}

```

### Security Settings

```

{
 "security": {
 "verify_ssl": true,
 "api_key_masking": true,
 "log_requests": false,
 "log_responses": false,
 "encrypt_history": false
 }
}

```

### Performance Tuning

```

{
 "performance": {
 "buffer_size": 8192,
 "chunk_size": 1024,
 "memory_limit": "100MB",
 "cache_responses": false,
 "preload_models": false
 }
}

```

## 1.9.10 Multiple Configurations

### Work vs Personal

Create separate configurations for different contexts:

**work-config.json:**

```

{
 "api_key": "sk-work-key-here",
 "provider_type": "openai",
 "default_model": "gpt-4",
 "ui": {
 "theme": "professional",

```

(continues on next page)

(continued from previous page)

```
"show_timestamps": true
},
"behavior": {
 "auto_save_history": true,
 "history_file": "~/.perspt/work_history.json"
}
}
```

personal-config.json:

```
{
 "api_key": "sk-personal-key-here",
 "provider_type": "anthropic",
 "default_model": "claude-3-sonnet-20240229",
 "ui": {
 "theme": "vibrant",
 "show_timestamps": false
 },
 "behavior": {
 "auto_save_history": false
 }
}
```

Usage:

```
Work configuration
perspt --config work-config.json

Personal configuration
perspt --config personal-config.json

Create aliases for convenience
alias work-ai="perspt --config ~/.perspt/work-config.json"
alias personal-ai="perspt --config ~/.perspt/personal-config.json"
```

## 1.9.11 Configuration Validation

Perspt validates your configuration and provides helpful error messages:

```
Validate configuration without starting
perspt --config config.json --validate

Check configuration and list available models
perspt --config config.json --list-models
```

Common validation errors:

- **Invalid API key format:** Ensure your API key follows the correct format
- **Missing required fields:** Some providers require specific configuration
- **Invalid model names:** Use `--list-models` to see available options
- **Network connectivity:** Check internet connection and proxy settings

### 1.9.12 Configuration Templates

Generate template configurations for different use cases:

```
Generate basic template
perspt --generate-config basic > config.json

Generate advanced template
perspt --generate-config advanced > advanced-config.json

Generate provider-specific template
perspt --generate-config openai > openai-config.json
```

### 1.9.13 Migration and Import

#### From Other Tools

Import configurations from similar tools:

```
Import from environment variables
perspt --import-env > config.json

Import from ChatGPT CLI config
perspt --import chatgpt-cli ~/.chatgpt-cli/config.yaml

Import from OpenAI CLI config
perspt --import openai-cli ~/.openai/config.json
```

#### Backup and Restore

```
Backup current configuration
cp ~/.config/perspt/config.json ~/.config/perspt/config.backup.json

Restore from backup
cp ~/.config/perspt/config.backup.json ~/.config/perspt/config.json

Export configuration with history
perspt --export-config --include-history > full-backup.json
```

### 1.9.14 Best Practices

#### Security

1. **Never commit API keys** to version control
2. **Use environment variables** for sensitive data
3. **Rotate API keys** regularly
4. **Use separate keys** for different projects
5. **Enable API key masking** in logs

## Organization

1. **Use descriptive config names** (work-config.json, research-config.json)
2. **Create aliases** for frequently used configurations
3. **Document your configurations** with comments (where supported)
4. **Use version control** for non-sensitive configuration parts
5. **Regular backups** of important configurations

## Performance

1. **Set appropriate timeouts** based on your network
2. **Configure retry settings** for reliability
3. **Use streaming** for better user experience
4. **Limit history length** to prevent memory issues
5. **Enable compression** for large chat histories

## 1.9.15 Troubleshooting

### Common Issues

#### Configuration not found:

```
Check current working directory
ls -la config.json

Check user config directory
ls -la ~/.config/perspt/

Use absolute path
perspt --config /full/path/to/config.json
```

#### Invalid JSON format:

```
Validate JSON syntax
cat config.json | python -m json.tool

Or use jq
jq . config.json
```

#### API key not working:

```
Test API key directly
curl -H "Authorization: Bearer $OPENAI_API_KEY" \
 "https://api.openai.com/v1/models"

Check environment variable
echo $OPENAI_API_KEY
```

#### Provider connection issues:

```
Test network connectivity
ping api.openai.com

Check proxy settings
echo $HTTP_PROXY $HTTPS_PROXY
```

(continues on next page)



(continued from previous page)

```
Test with verbose output
perspt --config config.json --verbose
```

## Getting Help

If you need assistance with configuration:

1. **Check the examples** in this guide
2. **Use the validation commands** to check your config
3. **Review the error messages** - they often contain helpful hints
4. **Ask the community** on [GitHub Discussions](#)
5. **File an issue** if you find a bug in configuration handling


### See also

- *Getting Started* - Basic setup and first run
- *AI Providers* - Provider-specific guides
- *Troubleshooting* - Common issues and solutions
- *Advanced Features* - Advanced usage patterns

## 1.10 Reference

Technical reference documentation.

☐ CLI Reference Complete command-line interface documentation.

*CLI Reference*  *Troubleshooting* Common issues and solutions.

*Troubleshooting* ☐ ☐ *Key Bindings* Keyboard shortcuts for TUI.

*Key Bindings*

### 1.10.1 CLI Reference

Complete command-line interface reference for Perspt.

#### Global Options

```
perspt [OPTIONS] <COMMAND>
```

| Option                                 | Description             |
|----------------------------------------|-------------------------|
| <code>-v, --verbose</code>             | Enable verbose logging  |
| <code>-c, --config &lt;FILE&gt;</code> | Configuration file path |
| <code>-h, --help</code>                | Print help information  |
| <code>-V, --version</code>             | Print version           |

## Commands Overview

| Command | Description                          |
|---------|--------------------------------------|
| chat    | Start interactive TUI chat session   |
| agent   | Run SRBN agent for autonomous coding |
| init    | Initialize project configuration     |
| config  | Manage configuration settings        |
| ledger  | Query and manage Merkle ledger       |
| status  | Show current agent status            |
| abort   | Abort current agent session          |
| resume  | Resume paused or crashed session     |

### perspt chat

**Usage:** perspt chat [OPTIONS]

Start an interactive TUI chat session.

|                                        |                              |
|----------------------------------------|------------------------------|
| <code>-m, --model &lt;MODEL&gt;</code> | Model to use (e.g., gpt-5.2) |
|----------------------------------------|------------------------------|

**Example:**

```
perspt chat
perspt chat --model claude-opus-4.5
```

### perspt agent

**Usage:** perspt agent [OPTIONS] <TASK>

Run the SRBN agent for autonomous coding.

**Arguments:**

|        |                                       |
|--------|---------------------------------------|
| <TASK> | Task description or path to task file |
|--------|---------------------------------------|

**Model Options:**

|                                           |                                     |
|-------------------------------------------|-------------------------------------|
| <code>--model &lt;MODEL&gt;</code>        | Override all model tiers            |
| <code>--architect-model &lt;M&gt;</code>  | Model for planning (deep reasoning) |
| <code>--actuator-model &lt;M&gt;</code>   | Model for code generation           |
| <code>--verifier-model &lt;M&gt;</code>   | Model for stability checking        |
| <code>--speculator-model &lt;M&gt;</code> | Model for fast lookahead            |

**Execution Options:**

|                                         |                                             |
|-----------------------------------------|---------------------------------------------|
| <code>-w, --workdir &lt;DIR&gt;</code>  | Working directory (default: current)        |
| <code>-y, --yes</code>                  | Auto-approve all actions                    |
| <code>--auto-approve-safe</code>        | Auto-approve read-only operations           |
| <code>-k, --complexity &lt;K&gt;</code> | Max complexity before approval (default: 5) |
| <code>--mode &lt;MODE&gt;</code>        | cautious, balanced, or yolo                 |

**SRBN Options:**

|                                                                          |                                           |
|--------------------------------------------------------------------------|-------------------------------------------|
| <code>--energy-weights &lt;<math>\alpha, \beta, \gamma</math>&gt;</code> | Lyapunov weights (default: 1.0, 0.5, 2.0) |
| <code>--stability-threshold &lt;<math>\epsilon</math>&gt;</code>         | Convergence threshold (default: 0.1)      |

**Limit Options:**

|                                     |                                         |
|-------------------------------------|-----------------------------------------|
| <code>--max-cost &lt;USD&gt;</code> | Maximum cost in dollars (0 = unlimited) |
| <code>--max-steps &lt;N&gt;</code>  | Maximum iterations (0 = unlimited)      |

**Examples:**

```
perspt agent "Create a calculator"
perspt agent -y -w ./project "Add tests"
perspt agent --architect-model gpt-5.2 --actuator-model claude-opus-4.5 "Build API"
```

**perspt init**

**Usage:** perspt init [OPTIONS]

Initialize project configuration.

|                       |                                       |
|-----------------------|---------------------------------------|
| <code>--memory</code> | Create PERSPT.md project memory file  |
| <code>--rules</code>  | Create .perspt/rules.star policy file |

**Example:**

```
perspt init --memory --rules
```

**perspt config**

**Usage:** perspt config [OPTIONS]

Manage configuration settings.

|                                      |                               |
|--------------------------------------|-------------------------------|
| <code>--show</code>                  | Display current configuration |
| <code>--set &lt;KEY=VALUE&gt;</code> | Set a configuration value     |
| <code>--edit</code>                  | Open in \$EDITOR              |

**Examples:**

```
perspt config --show
perspt config --set default.model=gpt-5.2
perspt config --edit
```

**perspt ledger**

**Usage:** perspt ledger [OPTIONS]

Query and manage the Merkle change ledger.

|                                      |                             |
|--------------------------------------|-----------------------------|
| <code>--recent</code>                | Show recent commits         |
| <code>--rollback &lt;HASH&gt;</code> | Rollback to specific commit |
| <code>--stats</code>                 | Show ledger statistics      |

### Examples:

```
perspt ledger --recent
perspt ledger --rollback abc123
perspt ledger --stats
```

### perspt status

**Usage:** `perspt status`

Show current agent session status. Displays:

- Session ID
- Current task
- Energy levels (V\_syn, V\_str, V\_log)
- Token usage

### perspt abort

**Usage:** `perspt abort [OPTIONS]`

Abort the current agent session.

|                          |                                  |
|--------------------------|----------------------------------|
| <code>-f, --force</code> | Force abort without confirmation |
|--------------------------|----------------------------------|

### perspt resume

**Usage:** `perspt resume [SESSION_ID]`

Resume a paused or crashed session.

|                           |                                                         |
|---------------------------|---------------------------------------------------------|
| <code>[SESSION_ID]</code> | Session ID to resume (optional, uses latest if omitted) |
|---------------------------|---------------------------------------------------------|

### Exit Codes

| Code | Meaning               |
|------|-----------------------|
| 0    | Success               |
| 1    | General error         |
| 2    | Configuration error   |
| 3    | Provider/API error    |
| 4    | Agent aborted by user |

### See Also

- *Configuration* - Configuration guide
- *Agent Options* - Agent options detail
- *perspt-cli API* - API documentation

## 1.10.2 Troubleshooting

Common issues and solutions.

### API Key Issues

**Error:** API key not found

```
Check if key is set
echo $OPENAI_API_KEY

Set the key
export OPENAI_API_KEY="sk-..."
```

**Error:** Invalid API key

- Verify the key is correct (no extra spaces)
- Check if the key has been revoked
- Regenerate from provider dashboard

### Provider Connection

**Error:** Connection refused

```
Check internet connectivity
curl https://api.openai.com

For Ollama, ensure it's running
ollama serve
```

**Error:** Rate limit exceeded

- Wait and retry
- Reduce request frequency
- Consider using a different provider

### Model Issues

**Error:** Model not found

```
List available models
perspt --list-models

For Ollama
ollama list
```

**Error:** Context length exceeded

- Use a model with longer context
- Reduce conversation history
- Clear chat and start fresh

### Agent Mode Issues

**Agent stuck in retry loop**

1. Check LSP is working:

```
ty check file.py
```

2. Lower stability threshold:

```
perspt agent --stability-threshold 0.5 "task"
```

3. Check for unsolvable errors in code

### High energy despite clean code

1. Run tests manually:

```
pytest -v
```

2. Check LSP diagnostics
3. Adjust energy weights:

```
perspt agent --energy-weights "0.5,0.5,1.0" "task"
```

### Agent aborted unexpectedly

```
Check session status
perspt status

Resume if possible
perspt resume
```

## TUI Issues

### Terminal rendering problems

- Ensure terminal supports 256 colors
- Try a different terminal emulator
- Check \$TERM environment variable

### Keyboard shortcuts not working

- Check terminal keybinding conflicts
- Try raw mode: `perspt --raw`

## Configuration Issues

### Config file not found

```
Create config directory
mkdir -p ~/.perspt

Initialize config
perspt config --edit
```

### Invalid config format

Check TOML syntax:

```
Validate TOML
python -c "import toml; toml.load(open('~/.perspt/config.toml', 'rb'))"
```

## Build Issues

### Compilation errors

```
Update Rust
rustup update

Clean and rebuild
cargo clean
cargo build --release
```

### Missing dependencies

```
macOS
brew install openssl pkg-config

Ubuntu/Debian
sudo apt install libssl-dev pkg-config
```

## Performance Issues

### Slow response times

- Try Groq for faster inference
- Use smaller models (e.g., gemini-3-flash)
- Check network latency

### High memory usage

- Reduce conversation history
- Use streaming mode
- Restart application periodically

## Debug Mode

Enable verbose logging:

```
perspt -v chat
perspt -v agent "task"

Maximum verbosity
RUST_LOG=debug perspt chat
```

## Getting Help

1. **Check documentation:** This book
2. **GitHub Issues:** [github.com/eonseed/perspt/issues](https://github.com/eonseed/perspt/issues)
3. **Logs:** Check ~/.perspt/logs/

## See Also

- *CLI Reference* - Command reference
- *Configuration* - Configuration guide

### 1.10.3 Key Bindings

Keyboard shortcuts for Perspt TUI.

#### Chat Mode

| Key                        | Action                         |
|----------------------------|--------------------------------|
| <b>Enter</b>               | Send current message           |
| <b>Esc</b>                 | Exit application               |
| <b>Ctrl+C</b>              | Force exit with cleanup        |
| <b>Ctrl+D</b>              | Exit (EOF)                     |
| <b>↑ / ↓</b>               | Scroll chat history            |
| <b>Page Up / Page Down</b> | Fast scroll (10 lines)         |
| <b>Home / End</b>          | Jump to top/bottom of history  |
| <b>Backspace</b>           | Delete character before cursor |
| <b>Delete</b>              | Delete character at cursor     |
| <b>← / →</b>               | Move cursor in input           |

#### Agent Mode (Dashboard)

| Key            | Action                |
|----------------|-----------------------|
| <b>q / Esc</b> | Exit application      |
| <b>Tab</b>     | Switch between panels |
| <b>↑ / k</b>   | Select previous item  |
| <b>↓ / j</b>   | Select next item      |
| <b>Enter</b>   | Expand/view details   |
| <b>Space</b>   | Toggle selection      |

#### Agent Mode (Review Modal)

| Key          | Action               |
|--------------|----------------------|
| <b>y</b>     | Approve change       |
| <b>n</b>     | Reject change        |
| <b>d</b>     | View diff            |
| <b>e</b>     | Edit before applying |
| <b>Esc</b>   | Cancel review        |
| <b>Enter</b> | Confirm action       |

#### Agent Mode (Diff Viewer)

| Key                        | Action            |
|----------------------------|-------------------|
| <b>↑ / k</b>               | Scroll up         |
| <b>↓ / j</b>               | Scroll down       |
| <b>Page Up / Page Down</b> | Fast scroll       |
| <b>g</b>                   | Jump to top       |
| <b>G</b>                   | Jump to bottom    |
| <b>q / Esc</b>             | Close diff viewer |



## Commands

Type these in the chat input:

| Command       | Action                             |
|---------------|------------------------------------|
| /save         | Save conversation (auto timestamp) |
| /save <file>  | Save to specific file              |
| /clear        | Clear conversation history         |
| /help         | Show available commands            |
| /model <name> | Switch model                       |
| /quit         | Exit application                   |

## Vim-style Navigation

For users who prefer vim bindings:

| Key | Action            |
|-----|-------------------|
| j   | Down              |
| k   | Up                |
| h   | Left (in input)   |
| l   | Right (in input)  |
| gg  | Top of history    |
| G   | Bottom of history |

## See Also

- *First Chat* - Getting started
- *CLI Reference* - CLI commands

## 1.11 API Reference

Complete API documentation for Perspt's 6-crate workspace architecture.

### 1.11.1 perspt-cli API

The command-line interface entry point with 8 subcommands.

#### Overview

`perspt-cli` provides the user-facing CLI with subcommand architecture:

- **chat** - Interactive TUI mode
- **agent** - SRBN autonomous coding
- **init** - Project initialization
- **config** - Configuration management
- **ledger** - Merkle ledger operations
- **status** - Agent status
- **abort** - Cancel session
- **resume** - Resume session

## Global Options

These options apply to all subcommands:

```
perspt [OPTIONS] <COMMAND>
```

### Options:

|                     |                         |
|---------------------|-------------------------|
| -v, --verbose       | Enable verbose logging  |
| -c, --config <FILE> | Configuration file path |
| -h, --help          | Print help              |
| -V, --version       | Print version           |

## Chat Command

Start an interactive TUI chat session.

```
perspt chat [OPTIONS]
```

### Options:

|                     |                              |
|---------------------|------------------------------|
| -m, --model <MODEL> | Model to use (e.g., gpt-5.2) |
|---------------------|------------------------------|

## Agent Command

Run the SRBN agent on a coding task.

```
perspt agent [OPTIONS] <TASK>
```

### Arguments:

|        |                                       |
|--------|---------------------------------------|
| <TASK> | Task description or path to task file |
|--------|---------------------------------------|

### Model Selection:

|                            |                                     |
|----------------------------|-------------------------------------|
| --model <MODEL>            | Override all model tiers            |
| --architect-model <MODEL>  | Model for planning (deep reasoning) |
| --actuator-model <MODEL>   | Model for code generation           |
| --verifier-model <MODEL>   | Model for stability checking        |
| --speculator-model <MODEL> | Model for fast lookahead            |

### Execution Control:

|                      |                                            |
|----------------------|--------------------------------------------|
| -w, --workdir <DIR>  | Working directory (default: current)       |
| -y, --yes            | Auto-approve all actions                   |
| --auto-approve-safe  | Auto-approve read-only operations          |
| -k, --complexity <K> | Max tasks before approval (default: 5)     |
| -m, --mode <MODE>    | Execution mode: cautious   balanced   yolo |

### SRBN Parameters:

|                                              |                                           |
|----------------------------------------------|-------------------------------------------|
| --energy-weights < $\alpha, \beta, \gamma$ > | Lyapunov weights (default: 1.0, 0.5, 2.0) |
| --stability-threshold < $\epsilon$ >         | Convergence threshold (default: 0.1)      |

### Limits:

|                  |                                         |
|------------------|-----------------------------------------|
| --max-cost <USD> | Maximum cost in dollars (0 = unlimited) |
| --max-steps <N>  | Maximum iterations (0 = unlimited)      |

## Examples

```
Basic task
perspt agent "Create a Python calculator"

With workspace directory
perspt agent -w /path/to/project "Add unit tests"

Auto-approve all
perspt agent -y "Refactor the parser"

Custom models
perspt agent --architect-model gpt-5.2 --actuator-model claude-opus-4.5 "Build API"

Custom energy weights
perspt agent --energy-weights "2.0,1.0,3.0" "Fix type errors"
```

## Init Command

Initialize project configuration and memory files.

```
perspt init [OPTIONS]

Options:
 --memory Create PERSPT.md project memory file
 --rules Create default Starlark policy rules (.perspt/rules.star)
```

## Examples

```
Initialize with memory file
perspt init --memory

Initialize with policy rules
perspt init --rules

Initialize both
perspt init --memory --rules
```

## Config Command

Manage configuration settings.

```
perspt config [OPTIONS]

Options:
 --show Show current configuration
 --set <KEY=VAL> Set a configuration value
 --edit Open in $EDITOR
```

## Examples

```
Show configuration
perspt config --show

Set default model
perspt config --set default_model=gpt-5.2

Edit in vim
EDITOR=vim perspt config --edit
```

## Ledger Command

Query and manage the Merkle ledger.

```
perspt ledger [OPTIONS]

Options:
 --recent Show recent commits
 --rollback <HASH> Rollback to a specific commit
 --stats Show ledger statistics
```

## Examples

```
View recent commits
perspt ledger --recent

Rollback to commit
perspt ledger --rollback abc123

Show statistics
perspt ledger --stats
```

## Status Command

Show current agent status.

```
perspt status
```

Displays: - Current session ID - Task in progress - Energy levels - Token usage

## Abort Command

Abort the current agent session.

```
perspt abort [OPTIONS]

Options:
 -f, --force Force abort without confirmation
```

## Resume Command

Resume a paused or crashed session.

```
perspt resume [SESSION_ID]
```

Arguments:

[SESSION\_ID] Session ID to resume (optional, uses latest if omitted)

## Implementation

### Command Routing

```
#[derive(Subcommand)]
enum Commands {
 Chat { model: Option<String> },
 Agent { task: String, ... },
 Init { memory: bool, rules: bool },
 Config { show: bool, set: Option<String>, edit: bool },
 Ledger { recent: bool, rollback: Option<String>, stats: bool },
 Status,
 Abort { force: bool },
 Resume { session_id: Option<String> },
}

#[tokio::main]
async fn main() -> Result<()> {
 let cli = Cli::parse();

 match cli.command {
 None | Some(Commands::Chat { .. }) => commands::chat::run().await,
 Some(Commands::Agent { .. }) => commands::agent::run(...).await,
 Some(Commands::Init { .. }) => commands::init::run(...).await,
 // etc.
 }
}
```

### Source Code

- crates/perspt-cli/src/main.rs - CLI definition and routing
- crates/perspt-cli/src/commands/ - Command implementations: - chat.rs - agent.rs - init.rs - config.rs - ledger.rs - status.rs - abort.rs - resume.rs

## 1.11.2 perspt-core API

The core crate providing LLM abstraction, configuration, and memory management.

### Overview

perspt-core contains the fundamental abstractions used by all other crates:

- **GenAIProvider** - Thread-safe LLM provider with streaming support
- **Config** - Simple configuration struct
- **Memory** - Conversation memory management

## GenAIProvider

Thread-safe LLM provider built on the `genai` crate with `Arc<RwLock>` for safe sharing across async tasks.

### Struct Definition

```
/// Thread-safe LLM provider implementation using Arc<RwLock>.
///
/// This provider can be cheaply cloned and shared across multiple agents.
/// Each clone shares the same underlying client and rate limiting state.
#[derive(Clone)]
pub struct GenAIProvider {
 /// The underlying genai client
 client: Arc<Client>,
 /// Shared state for rate limiting and metrics
 shared: Arc<RwLock<SharedState>>,
}

struct SharedState {
 total_tokens_used: usize,
 request_count: usize,
}
```

### Constructor Methods

```
impl GenAIProvider {
 /// Creates a new GenAI provider with automatic configuration.
 ///
 /// Uses genai's default client which auto-detects API keys
 /// from environment variables.
 pub fn new() -> Result<Self>

 /// Creates a new GenAI provider with explicit configuration.
 ///
 /// # Arguments
 /// * `provider_type` - Provider name: "openai", "anthropic", "gemini", etc.
 /// * `api_key` - API key for the provider
 ///
 /// Sets the appropriate environment variable before creating the client.
 pub fn new_with_config(
 provider_type: Option<&str>,
 api_key: Option<&str>
) -> Result<Self>
}
```

### Streaming Response

```
impl GenAIProvider {
 /// Generates a streaming response to a channel.
 ///
 /// Sends tokens as they arrive via the provided mpsc sender.
 /// Sends EOT_SIGNAL ("<|EOT|>") when complete.
```

(continues on next page)

(continued from previous page)

```

///
/// # Arguments
/// * `model` - Model identifier (e.g., "gpt-5.2", "claude-opus-4.5")
/// * `messages` - Conversation history as ChatMessage vec
/// * `sender` - Channel to send streaming tokens
pub async fn generate_response_stream_to_channel(
 &self,
 model: &str,
 messages: Vec<ChatMessage>,
 sender: mpsc::Sender<String>,
) -> Result<()>
}

```

### Metrics Methods

```

impl GenAIProvider {
 /// Get total tokens used across all requests
 pub async fn get_total_tokens_used(&self) -> usize

 /// Get total request count
 pub async fn get_request_count(&self) -> usize
}

```

### Supported Providers

The provider type maps to environment variables:

| Provider  | Environment Variable |
|-----------|----------------------|
| openai    | OPENAI_API_KEY       |
| anthropic | ANTHROPIC_API_KEY    |
| gemini    | GEMINI_API_KEY       |
| groq      | GROQ_API_KEY         |
| cohere    | COHERE_API_KEY       |
| xai       | XAI_API_KEY          |
| deepseek  | DEEPSEEK_API_KEY     |
| ollama    | (none - local)       |

### Config

Simple configuration struct:

```

/// Main configuration struct
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct Config {
 pub provider: String,
 pub model: String,
 pub api_key: Option<String>,
}

impl Default for Config {

```

(continues on next page)

(continued from previous page)

```
fn default() -> Self {
 Self {
 provider: "openai".to_string(),
 model: "gpt-4".to_string(),
 api_key: None,
 }
}
```

## Memory

Conversation memory management for context handling.

## Usage Example

```
use perspt_core::llm_provider::GenAIProvider;
use tokio::sync::mpsc;

#[tokio::main]
async fn main() -> Result<()> {
 // Create provider with auto-detection
 let provider = GenAIProvider::new()?;

 // Create channel for streaming
 let (tx, mut rx) = mpsc::channel(100);

 // Start streaming
 tokio::spawn(async move {
 while let Some(token) = rx.recv().await {
 if token == "<|EOT|>" { break; }
 print!("{}", token);
 }
 });

 // Generate response
 provider.generate_response_stream_to_channel(
 "gpt-5.2",
 vec![ChatMessage::user("Hello!")],
 tx,
).await?;

 Ok(())
}
```

## Source Code

- crates/perspt-core/src/lib.rs
- crates/perspt-core/src/llm\_provider.rs
- crates/perspt-core/src/config.rs
- crates/perspt-core/src/memory.rs



### 1.11.3 perspt-agent API

The SRBN (Stabilized Recursive Barrier Network) engine for autonomous coding.

#### Overview

perspt-agent implements the core autonomous coding capabilities:

- **SRBNOrchestrator** - Main control loop for task execution
- **LspClient** - Language Server Protocol integration
- **AgentTools** - File and shell operations
- **PythonTestRunner** - pytest integration with V\_log calculation
- **MerkleLedger** - Change tracking with integrity verification

#### SRBN Control Loop

The orchestrator follows the SRBN algorithm:

#### SRBNOrchestrator

The main orchestrator class:

```
pub struct SRBNOrchestrator {
 provider: Arc<GenAIProvider>,
 workspace: PathBuf,
 lsp_client: Option<LspClient>,
 test_runner: Option<PythonTestRunner>,
 ledger: MerkleLedger,
 tools: AgentTools,

 // Model configuration
 architect_model: String,
 actuator_model: String,
 verifier_model: String,
 speculator_model: String,

 // Energy weights
 alpha: f32, // V_syn weight (default: 1.0)
 beta: f32, // V_str weight (default: 0.5)
 gamma: f32, // V_log weight (default: 2.0)

 // Convergence threshold
 epsilon: f32, // Default: 0.1
}
```

#### Constructor

```
impl SRBNOrchestrator {
 pub async fn new(
 provider: Arc<GenAIProvider>,
 workspace: PathBuf,
 options: OrchestratorOptions,
) -> Result<Self>
}
```

(continues on next page)

(continued from previous page)

```
pub struct OrchestratorOptions {
 pub architect_model: Option<String>,
 pub actuator_model: Option<String>,
 pub verifier_model: Option<String>,
 pub speculator_model: Option<String>,
 pub alpha: f32,
 pub beta: f32,
 pub gamma: f32,
 pub epsilon: f32,
 pub max_retries_compile: usize, // Default: 3
 pub max_retries_tool: usize, // Default: 5
}
```

## Main Execution

```
impl SRBNOrchestrator {
 /// Execute a task through the SRBN loop
 pub async fn execute(&mut self, task: &str) -> Result<ExecutionResult>

 /// Execute with approval callback for complexity > K
 pub async fn execute_with_approval<F>(
 &mut self,
 task: &str,
 complexity_k: usize,
 approval_fn: F,
) -> Result<ExecutionResult>
 where
 F: Fn(&TaskPlan) -> bool
}
```

## Energy Computation

Lyapunov Energy  $V(x)$ :

$$V(x) = \alpha \cdot V_{syn} + \beta \cdot V_{str} + \gamma \cdot V_{log}$$

Components:

- **V\_syn** - Syntax energy from LSP diagnostics
- **V\_str** - Structural energy from code analysis
- **V\_log** - Logic energy from test failures

```
pub struct Energy {
 pub v_syn: f32,
 pub v_str: f32,
 pub v_log: f32,
 pub total: f32,
}

impl Energy {
 pub fn compute(
```

(continues on next page)

(continued from previous page)

```

 lsp_diagnostics: &[Diagnostic],
 test_results: &TestResults,
 alpha: f32,
 beta: f32,
 gamma: f32,
) -> Self
}

```

## Types

### TaskPlan

```

/// JSON-serializable task decomposition
#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct TaskPlan {
 pub nodes: Vec<TaskNode>,
 pub dependencies: Vec<(usize, usize)>,
}

#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct TaskNode {
 pub id: usize,
 pub description: String,
 pub node_type: NodeType,
 pub status: NodeStatus,
 pub files_affected: Vec<String>,
}

pub enum NodeType {
 Create,
 Modify,
 Delete,
 Test,
 Shell,
}

pub enum NodeStatus {
 Pending,
 InProgress,
 Completed,
 Failed(String),
}

```

### ToolCall

```

#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct ToolCall {
 pub name: String,
 pub arguments: serde_json::Value,
}

```

(continues on next page)

(continued from previous page)

```
pub enum ToolResult {
 Success(String),
 Error(String),
}
```

## LspClient

Language Server Protocol client for real-time diagnostics:

```
pub struct LspClient {
 process: Child,
 reader: BufReader<ChildStdout>,
 writer: BufWriter<ChildStdin>,
}

impl LspClient {
 /// Start LSP server for Python using `ty`
 pub async fn new_python(workspace: &Path) -> Result<Self>

 /// Get diagnostics for a file
 pub async fn get_diagnostics(&mut self, file: &Path) -> Result<Vec<Diagnostic>>

 /// Notify file change
 pub async fn did_change(&mut self, file: &Path, content: &str) -> Result<()>
}

pub struct Diagnostic {
 pub severity: DiagnosticSeverity,
 pub message: String,
 pub range: Range,
}

pub enum DiagnosticSeverity {
 Error,
 Warning,
 Information,
 Hint,
}
```

## PythonTestRunner

pytest integration with V\_log calculation:

```
pub struct PythonTestRunner {
 workspace: PathBuf,
}

impl PythonTestRunner {
 pub fn new(workspace: PathBuf) -> Self

 /// Run pytest and compute V_log
 pub async fn run(&self) -> Result<TestResults>
}
```

(continues on next page)

(continued from previous page)

```

}

pub struct TestResults {
 pub passed: usize,
 pub failed: usize,
 pub errors: usize,
 pub v_log: f32,
 pub failures: Vec<TestFailure>,
}

pub struct TestFailure {
 pub test_name: String,
 pub message: String,
 pub criticality: f32, // Weight for V_log
}

```

## AgentTools

Available tools for the agent:

| Tool            | Description                   |
|-----------------|-------------------------------|
| read_file       | Read file contents            |
| write_file      | Write/create file             |
| search_files    | Search for patterns in files  |
| list_directory  | List directory contents       |
| execute_shell   | Run shell command (sandboxed) |
| get_diagnostics | Get LSP diagnostics           |
| run_tests       | Execute pytest                |

```

pub struct AgentTools {
 workspace: PathBuf,
 policy_engine: Arc<PolicyEngine>,
 sandbox: SandboxedCommand,
}

impl AgentTools {
 pub fn available_tools() -> Vec<ToolDefinition>
 pub async fn execute(&self, call: &ToolCall) -> Result<ToolResult>
}

```

## MerkleLedger

Git-style change tracking with Merkle tree:

```

pub struct MerkleLedger {
 root: Option<Hash>,
 commits: Vec<Commit>,
}

impl MerkleLedger {

```

(continues on next page)

(continued from previous page)

```

pub fn new() -> Self

/// Record a commit
pub fn commit(&mut self, changes: Vec<Change>) -> Hash

/// Get commit by hash
pub fn get(&self, hash: &Hash) -> Option<&Commit>

/// Rollback to previous commit
pub fn rollback(&mut self, hash: &Hash) -> Result<()>
}

```

### Retry Policy

PSP-4 compliant retry limits:

| Error Type         | Max Retries | Action on Exhaustion |
|--------------------|-------------|----------------------|
| Compilation errors | 3           | Escalate to user     |
| Tool failures      | 5           | Escalate to user     |
| Review rejections  | 3           | Escalate to user     |

### Source Code

- crates/perspt-agent/src/orchestrator.rs (34KB)
- crates/perspt-agent/src/lsp.rs (28KB)
- crates/perspt-agent/src/tools.rs (12KB)
- crates/perspt-agent/src/types.rs (24KB)
- crates/perspt-agent/src/ledger.rs (6KB)
- crates/perspt-agent/src/test\_runner.rs (15KB)

## 1.11.4 perspt-tui API

Terminal UI components for Perspt built on the Ratatui framework.

### Overview

perspt-tui provides the visual interface for both chat and agent modes:

### Modules

#### AgentApp

The main TUI application for agent mode.

Listing 1: AgentApp structure

```

pub struct AgentApp {
 // Current view state
 view: View,
 // Task tree widget
 task_tree: TaskTree,
 // Status dashboard

```

(continues on next page)

(continued from previous page)

```

 dashboard: Dashboard,
 // Active diff viewer (if showing changes)
 diff_viewer: Option<DiffViewer>,
 // Review modal (if awaiting approval)
 review_modal: Option<ReviewModal>,
}

pub enum View {
 Dashboard,
 TaskTree,
 DiffView,
 Review,
}

impl AgentApp {
 pub fn new() -> Self
 pub fn run(&mut self, terminal: &mut Terminal<impl Backend>) -> Result<()>
 pub fn handle_event(&mut self, event: Event) -> Result<Option<Action>>
}

```

**run\_agent\_tui**(orchestrator: *SRBNOrchestrator*) → Result<()>

Entry point for the agent TUI. Initializes the terminal and runs the event loop.

## Dashboard

Status dashboard displaying metrics and progress.

Table 1: Dashboard Widgets

| Widget        | Information Displayed                                        |
|---------------|--------------------------------------------------------------|
| Session Info  | Session ID, start time, elapsed duration                     |
| Token Usage   | Input/output tokens, cost estimate                           |
| Energy Gauge  | Current $V(x)$ with $\alpha$ , $\beta$ , $\gamma$ components |
| Task Progress | Completed/total nodes, current node                          |

Listing 2: Dashboard structure

```

pub struct Dashboard {
 session_id: String,
 start_time: Instant,
 tokens_used: usize,
 current_energy: Energy,
 task_progress: (usize, usize),
}

impl Dashboard {
 pub fn render(&self, frame: &mut Frame, area: Rect)
 pub fn update_energy(&mut self, energy: Energy)
 pub fn update_tokens(&mut self, tokens: usize)
}

```

## TaskTree

Hierarchical task visualization.

Listing 3: TaskTree widget

```
pub struct TaskTree {
 nodes: Vec<TreeNode>,
 selected: usize,
 expanded: HashSet<usize>,
}

pub struct TreeNode {
 pub id: usize,
 pub label: String,
 pub status: NodeStatus,
 pub children: Vec<usize>,
}

impl TaskTree {
 pub fn from_task_plan(plan: &TaskPlan) -> Self
 pub fn render(&self, frame: &mut Frame, area: Rect)
 pub fn select_next(&mut self)
 pub fn select_prev(&mut self)
 pub fn toggle_expand(&mut self)
}
```

## DiffViewer

Side-by-side file diff display.

Listing 4: DiffViewer widget

```
pub struct DiffViewer {
 file_path: PathBuf,
 old_content: String,
 new_content: String,
 scroll_offset: usize,
}

impl DiffViewer {
 pub fn new(path: PathBuf, old: String, new: String) -> Self
 pub fn render(&self, frame: &mut Frame, area: Rect)
 pub fn scroll_up(&mut self)
 pub fn scroll_down(&mut self)
}
```

## ReviewModal

Change approval/rejection modal.



Listing 5: ReviewModal widget

```
pub struct ReviewModal {
 changes: Vec<Change>,
 selected: usize,
}

pub enum ReviewDecision {
 Approve,
 Reject,
 Edit,
}

impl ReviewModal {
 pub fn new(changes: Vec<Change>) -> Self
 pub fn render(&self, frame: &mut Frame, area: Rect)
 pub fn get_decision(&self) -> Option<ReviewDecision>
}
```

### Key Bindings

| Key     | Dashboard/Tree View | Diff/Review View |
|---------|---------------------|------------------|
| q / Esc | Exit application    | Close modal      |
| ↑ / k   | Select previous     | Scroll up        |
| ↓ / j   | Select next         | Scroll down      |
| Enter   | Expand/view details | Confirm action   |
| Tab     | Switch view         | Switch pane      |
| y       | —                   | Approve change   |
| n       | —                   | Reject change    |

### Source Code

crates/perspt-tui/src/agent\_app.rs: Main application (7KB) crates/perspt-tui/src/dashboard.rs: Status dashboard (8KB) crates/perspt-tui/src/diff\_viewer.rs: Diff display (6KB) crates/perspt-tui/src/review\_modal.rs: Review UI (6KB) crates/perspt-tui/src/task\_tree.rs: Task hierarchy (4KB)

## 1.11.5 perspt-policy API

Security policy engine for command approval and sanitization.

### Overview

perspt-policy provides security controls for agent operations:

### PolicyEngine

Starlark-based policy evaluation engine.

Listing 6: PolicyEngine structure

```

pub struct PolicyEngine {
 rules: Vec<Rule>,
 default_action: Action,
}

pub enum Action {
 Allow,
 Deny,
 Prompt,
}

pub struct Rule {
 pub pattern: String,
 pub action: Action,
 pub reason: Option<String>,
}

impl PolicyEngine {
 /// Create engine with default rules
 pub fn new() -> Self

 /// Load rules from Starlark file
 pub fn load_rules(path: &Path) -> Result<Self>

 /// Evaluate a command against rules
 pub fn evaluate(&self, command: &str) -> PolicyDecision
}

pub struct PolicyDecision {
 pub action: Action,
 pub matched_rule: Option<Rule>,
 pub reason: String,
}

```

### Default Rules

The engine includes built-in safety rules:

| Pattern     | Action | Reason                    |
|-------------|--------|---------------------------|
| rm -rf /    | Deny   | Destructive root deletion |
| rm -rf ~    | Deny   | Home directory deletion   |
| chmod 777   | Prompt | Insecure permissions      |
| curl   bash | Deny   | Remote code execution     |
| sudo *      | Prompt | Privilege escalation      |

### Sanitizer

Command input sanitization and validation.

Listing 7: Sanitizer structure

```

pub struct Sanitizer {
 // Sanitization rules
}

impl Sanitizer {
 pub fn new() -> Self

 /// Sanitize a command string
 pub fn sanitize(&self, command: &str) -> Result<String>

 /// Check for path traversal attempts
 pub fn check_path_traversal(&self, path: &str) -> bool

 /// Check for command injection
 pub fn check_injection(&self, input: &str) -> bool
}

```

### Security Checks

| Check                | Description                                  |
|----------------------|----------------------------------------------|
| Path Traversal       | Detects ../ patterns escaping workspace      |
| Command Injection    | Detects ;,  , &&, \$( ) in untrusted input   |
| Null Bytes           | Removes null bytes that can truncate strings |
| Shell Metacharacters | Escapes or rejects dangerous characters      |

### Custom Rules

Create custom Starlark rules in .perspt/rules.star:

Listing 8: Example rules.star

```
Allow read operations
allow("cat *")
allow("head *")
allow("tail *")

Prompt for writes
prompt("rm *", reason="File deletion")
prompt("mv *", reason="File move/rename")

Deny dangerous operations
deny("rm -rf *", reason="Recursive force delete")
deny("chmod -R *", reason="Recursive permission change")
```

## Usage Example

Listing 9: Using PolicyEngine

```
use perspt_policy::{PolicyEngine, Sanitizer, Action};

fn check_command(cmd: &str) -> Result<()> {
 let sanitizer = Sanitizer::new();
 let engine = PolicyEngine::new();

 // Sanitize input
 let clean_cmd = sanitizer.sanitize(cmd)?;

 // Evaluate policy
 let decision = engine.evaluate(&clean_cmd);

 match decision.action {
 Action::Allow => execute(clean_cmd),
 Action::Prompt => {
 if user_approves(&decision.reason) {
 execute(clean_cmd)
 }
 },
 Action::Deny => {
 eprintln!("Denied: {}", decision.reason);
 }
 }
}
```

## Source Code

crates/perspt-policy/src/engine.rs: Policy engine (7KB) crates/perspt-policy/src/sanitize.rs: Sanitizer (5KB)

### 1.11.6 perspt-sandbox API

Process isolation for safe command execution.

#### Overview

perspt-sandbox provides controlled execution of shell commands with resource limits and isolation.

#### SandboxedCommand

Safe command execution with limits.

Listing 10: SandboxedCommand structure

```
pub struct SandboxedCommand {
 command: String,
 args: Vec<String>,
 working_dir: PathBuf,
 timeout: Duration,
 max_output: usize,
}

pub struct CommandResult {
 pub exit_code: i32,
 pub stdout: String,
 pub stderr: String,
 pub duration: Duration,
 pub truncated: bool,
}

impl SandboxedCommand {
 /// Create a new sandboxed command
 pub fn new(command: &str) -> Self

 /// Set working directory
 pub fn working_dir(mut self, path: PathBuf) -> Self

 /// Set execution timeout
 pub fn timeout(mut self, duration: Duration) -> Self

 /// Set maximum output size
 pub fn max_output(mut self, bytes: usize) -> Self

 /// Add command arguments
 pub fn args<I, S>(mut self, args: I) -> Self
 where
 I: IntoIterator<Item = S>,
 S: AsRef<str>

 /// Execute the command
 pub async fn execute(self) -> Result<CommandResult>
}
```

## Default Limits

| Resource         | Default Limit | Purpose                     |
|------------------|---------------|-----------------------------|
| Timeout          | 60 seconds    | Prevent hanging processes   |
| Output Size      | 1 MB          | Prevent memory exhaustion   |
| Process Count    | 10            | Limit fork bombs            |
| File Descriptors | 256           | Prevent resource exhaustion |

## Security Features

### Isolation Mechanisms

- **Working Directory Restriction:** Commands run in specified workspace only
- **Environment Sanitization:** Only safe environment variables passed
- **Output Truncation:** Large outputs are truncated with warning
- **Timeout Enforcement:** Processes killed after timeout

## Usage Example

Listing 11: Using SandboxedCommand

```
use perspt_sandbox::SandboxedCommand;
use std::time::Duration;

#[tokio::main]
async fn main() -> Result<()> {
 let result = SandboxedCommand::new("pytest")
 .args(["tests/", "-v"])
 .working_dir("/path/to/project".into())
 .timeout(Duration::from_secs(120))
 .max_output(2 * 1024 * 1024) // 2MB
 .execute()
 .await?;

 println!("Exit code: {}", result.exit_code);
 println!("Duration: {:?}", result.duration);

 if result.truncated {
 println!("Warning: Output was truncated");
 }

 println!("{}", result.stdout);

 Ok(())
}
```

Integration with Agent

The agent uses SandboxedCommand for all shell operations:

Listing 12: Agent tool integration

```
impl AgentTools {
 async fn execute_shell(&self, cmd: &str) -> Result<ToolResult> {
 // First, check policy
 let decision = self.policy_engine.evaluate(cmd);
 if decision.action == Action::Deny {
 return Err(anyhow!("Denied: {}", decision.reason));
 }

 // Execute in sandbox
 let result = SandboxedCommand::new("sh")
 .args(["-c", cmd])
 .working_dir(self.workspace.clone())
 .timeout(Duration::from_secs(60))
 .execute()
 .await?;

 Ok(ToolResult::Success(result.stdout))
 }
}
```





Source Code

crates/perspt-sandbox/src/command.rs: SandboxedCommand (5KB)

1.11.7 Crate Summary

| Crate                     | Description                        | Key Types                          |
|---------------------------|------------------------------------|------------------------------------|
| <i>perspt-cli API</i>     | CLI entry point with 8 subcommands | Commands, Cli                      |
| <i>perspt-core API</i>    | LLM provider, config, memory       | GenAIProvider, Config              |
| <i>perspt-agent API</i>   | SRBN engine for autonomous coding  | SRBNOrchestrator, TaskPlan, Energy |
| <i>perspt-tui API</i>     | Terminal UI components             | AgentApp, Dashboard, DiffViewer    |
| <i>perspt-policy API</i>  | Security policy engine             | PolicyEngine, Sanitizer            |
| <i>perspt-sandbox API</i> | Process isolation                  | SandboxedCommand                   |

1.11.8 Architecture Quick Reference

- ☐ ☐ perspt-cli    **8 Subcommands:** chat, agent, init, config, ledger, status, abort, resume
- perspt-cli API*                      ☐ perspt-core    **Thread-safe LLM:** GenAIProvider with Arc<RwLock>
- perspt-core API*                     perspt-agent    **SRBN Engine:** Orchestrator, LSP, Tools, Ledger
- perspt-agent API*                    perspt-tui    **Ratatui UI:** Dashboard, DiffViewer, ReviewModal
- perspt-tui API*                     ☐ perspt-policy    **Security:** Starlark rules, command sanitization
- perspt-policy API*                    perspt-sandbox    **Isolation:** Resource limits, process control

*perspt-sandbox API*

### 1.11.9 Common Patterns

#### Error Handling

All crates use `anyhow::Result` for error propagation:

```
use anyhow::{Context, Result};

fn example() -> Result<> {
 do_something()
 .context("Failed to do something")?;
 Ok(())
}
```

#### Async Operations

Built on Tokio async runtime:

```
use tokio::sync::mpsc;

async fn stream_response(sender: mpsc::Sender<String>) -> Result<> {
 // Stream tokens as they arrive
}
```

#### Thread-Safe Sharing

Use Arc for sharing across tasks:

```
use std::sync::Arc;

let provider = Arc::new(GenAIProvider::new()?);
let provider_clone = Arc::clone(&provider);

tokio::spawn(async move {
 provider_clone.generate_response(...).await
});
```


### 1.11.10 See Also


- *Architecture* - Workspace architecture overview
- *Contributing* - How to contribute
- *Testing* - Testing guide

## 1.12 Developer Guide

Guide for extending and contributing to Perspt.

  Architecture 6-crate workspace design and SRBN engine.

*Architecture*  Contributing How to contribute to Perspt.

*Contributing*  Extending Add commands, tools, and providers.



Extending Perspt      □ Testing    Testing strategies and tools.

Testing

1.12.1    Architecture

Perspt is built as a modern, modular Rust application using a **6-crate workspace architecture**. This design enables clean separation of concerns, independent testing, and easy extensibility.

Workspace Overview

Crate Dependency Graph

SRBN Control Flow

Crate Details

**perspt-cli**

The command-line interface providing 8 subcommands:

Table 2: CLI Subcommands

| Command | Purpose                | Key Options                                                     |
|---------|------------------------|-----------------------------------------------------------------|
| chat    | Interactive TUI        | --model <MODEL>                                                 |
| agent   | SRBN autonomous coding | --architect-model, --actuator-model<br>--energy-weights, --mode |
| init    | Project setup          | --memory, --rules                                               |
| config  | Configuration          | --show, --set, --edit                                           |
| ledger  | Merkle ledger          | --recent, --rollback, --stats                                   |
| status  | Agent status           | (none)                                                          |
| abort   | Cancel session         | --force                                                         |
| resume  | Resume session         | [SESSION_ID]                                                    |

Source: crates/perspt-cli/src/

**perspt-core**

Thread-safe LLM provider and configuration:

Listing 13: GenAIProvider - Thread-safe LLM abstraction

```
/// Thread-safe LLM provider using Arc<RwLock>.
/// Can be safely cloned and shared across async tasks.
#[derive(Clone)]
pub struct GenAIProvider {
 client: Arc<Client>,
 shared: Arc<RwLock<SharedState>>,
}

impl GenAIProvider {
 pub fn new() -> Result<Self>
```

(continues on next page)

(continued from previous page)

```

pub fn new_with_config(provider: Option<&str>, api_key: Option<&str>) -> Result<Self>
pub async fn generate_response_stream_to_channel(...) -> Result<()>
pub async fn get_total_tokens_used(&self) -> usize
}

```

**Modules:****config.rs**

Simple Config struct (provider, model, api\_key)

**llm\_provider.rs**

GenAIProvider with streaming support

**memory.rs**

Conversation memory management

**Source:** crates/perspt-core/src/**perspt-agent**

The Stabilized Recursive Barrier Network implementation.

**i Energy Computation**

$$V(x) = \alpha \cdot V_{syn} + \beta \cdot V_{str} + \gamma \cdot V_{log}$$

**Default weights:**  $\alpha=1.0$ ,  $\beta=0.5$ ,  $\gamma=2.0$ **Key Modules:**

| Module               | Size | Description                                  |
|----------------------|------|----------------------------------------------|
| orchestrator.rs      | 34KB | SRBN control loop, model tiers, retry policy |
| lsp.rs               | 28KB | LSP client for Python (ty server)            |
| tools.rs             | 12KB | Agent tools (search, read, write, shell)     |
| types.rs             | 24KB | TaskPlan, Node, Energy, ToolCall types       |
| ledger.rs            | 6KB  | Merkle ledger for change tracking            |
| test_runner.rs       | 15KB | pytest integration, V_log calculation        |
| context_retriever.rs | 10KB | Code context extraction                      |

**Source:** crates/perspt-agent/src/**perspt-tui**

Ratatui-based terminal interface components:

**agent\_app.rs**

Main agent mode TUI application

**dashboard.rs**

Status dashboard with metrics

**diff\_viewer.rs**

Side-by-side file diff display

**review\_modal.rs**

Change approval/rejection UI

**task\_tree.rs**

Hierarchical task visualization

**Source:** crates/perspt-tui/src/**perspt-policy**

Starlark-based policy engine for command approval:

Listing 14: Security policy engine

```
pub struct PolicyEngine {
 // Evaluates Starlark rules for command safety
}

pub struct Sanitizer {
 // Cleans and validates shell commands
 // Prevents path traversal, injection attacks
}
```

**Source:** crates/perspt-policy/src/**perspt-sandbox**

Safe command execution with process isolation.

**Source:** crates/perspt-sandbox/src/**Design Principles**

□ Modularity Each crate has a single responsibility:

- **perspt-cli** knows CLI, not LLM internals
- **perspt-core** provides LLM abstraction, not UI
- **perspt-agent** implements SRBN, delegates UI



Thread Safety GenAIProvider uses Arc&lt;RwLock&lt;SharedState&gt;&gt; for:

- Safe cloning across async tasks
- Shared token counting and rate limiting
- Concurrent access from orchestrator and UI

□□ Error Handling All crates use anyhow::Result for:

- Contextual error messages
- Error propagation with backtrace
- User-friendly error display



Async Architecture Built on Tokio runtime with:

- Streaming LLM responses via channels
- Non-blocking UI updates
- Concurrent tool execution

## Configuration Sources

Table 3: Configuration Priority (highest first)

| Priority | Source                | Example                                     |
|----------|-----------------------|---------------------------------------------|
| 1        | CLI Arguments         | <code>perspt agent --model gpt-5.2</code>   |
| 2        | Environment Variables | <code>OPENAI_API_KEY=sk-...</code>          |
| 3        | Config File           | <code>~/.perspt/config.toml</code>          |
| 4        | Built-in Defaults     | <code>provider: openai, model: gpt-4</code> |

## Supported Providers

| Provider  | Environment Variable           | Models                       |
|-----------|--------------------------------|------------------------------|
| OpenAI    | <code>OPENAI_API_KEY</code>    | GPT-5.2, o3-mini, o1-preview |
| Anthropic | <code>ANTHROPIC_API_KEY</code> | Claude Opus 4.5              |
| Google    | <code>GEMINI_API_KEY</code>    | Gemini 3 Flash/Pro           |
| Groq      | <code>GROQ_API_KEY</code>      | Llama 3.x                    |
| Ollama    | <i>(none)</i>                  | Local models                 |

## Extension Points

### Adding a New Command

1. Create `crates/perspt-cli/src/commands/mycommand.rs`
2. Add variant to `Commands` enum in `main.rs`
3. Add match arm in `main()`

### Adding a New Tool

1. Add tool definition to `crates/perspt-agent/src/tools.rs`
2. Register in `AgentTools::available_tools()`
3. Implement execution in `execute_tool()`

### Adding a Provider

The `genai` crate handles providers. To customize:

1. Set appropriate environment variable
2. Use provider-specific model names

#### See also

- *API Reference* - Per-crate API reference
- *Contributing* - How to contribute
- *Testing* - Testing guide

### 1.12.2 Contributing

How to contribute to Perspt.

#### Getting Started

1. **Fork** the repository on GitHub
2. **Clone** your fork:

```
git clone https://github.com/YOUR_USERNAME/perspt.git
cd perspt
```

3. **Create a branch:**

```
git checkout -b feat/your-feature
```

#### Development Setup

```
Install Rust (if needed)
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

Build
cargo build

Run tests
cargo test

Run with debug
cargo run -- chat
```

#### Code Style

- **Rust:** Follow rustfmt and clippy
- **Commit messages:** Conventional Commits format
- **Documentation:** Update docs for API changes

```
Format code
cargo fmt

Run linter
cargo clippy --all-targets
```

#### Commit Messages

Use [Conventional Commits](#):

```
feat: add new agent tool
fix: correct energy calculation
docs: update SRBN documentation
refactor: simplify orchestrator
test: add integration tests
chore: update dependencies
```

### Pull Request Process

1. Ensure all tests pass:

```
cargo test --all
```

2. Update documentation if needed
3. Create a PR with clear description
4. Address review feedback
5. Squash and merge when approved

### PSP Process

For significant changes, create a PSP (Perspt Specification Proposal):

1. Get PSP number from maintainers
2. Create docs/psps/source/psp-00000N.rst
3. Submit for review
4. Implement after acceptance

See *PSP: Perspt Specification Proposals* for details.

### Crate Structure

```
crates/
├── perspt-cli/ # CLI entry point
├── perspt-core/ # LLM provider
├── perspt-tui/ # Terminal UI
├── perspt-agent/ # SRBN engine
├── perspt-policy/ # Security
└── perspt-sandbox/ # Isolation
```

When contributing, add to the appropriate crate.

### See Also

- *Architecture* - Crate design
- *Testing* - Testing guide
- *PSP: Perspt Specification Proposals* - PSP process

## 1.12.3 Extending Perspt

How to add new capabilities to Perspt.

### Adding a New CLI Command

1. **Create command file:** crates/perspt-cli/src/commands/mycommand.rs

```
use anyhow::Result;

pub async fn run(arg: String) -> Result<()> {
 println!("Running mycommand with: {}", arg);
 Ok(())
}
```

2. **Register in mod.rs:** crates/perspt-cli/src/commands/mod.rs

```
pub mod mycommand;
```

3. Add to CLI enum: crates/perspt-cli/src/main.rs

```
#[derive(Subcommand)]
enum Commands {
 // ... existing commands

 /// My new command
 Mycommand {
 /// Argument
 arg: String,
 },
}
```

4. Add match arm:

```
Some(Commands::Mycommand { arg }) => commands::mycommand::run(arg).await,
```

### Adding a New Agent Tool

1. Define tool in tools.rs: crates/perspt-agent/src/tools.rs

```
pub fn available_tools() -> Vec<ToolDefinition> {
 vec![
 // ... existing tools
 ToolDefinition {
 name: "my_tool".to_string(),
 description: "Does something useful".to_string(),
 parameters: json!({
 "type": "object",
 "properties": {
 "input": { "type": "string", "description": "The input" }
 },
 "required": ["input"]
 }),
 },
]
}
```

2. Implement execution:

```
pub async fn execute(&self, call: &ToolCall) -> Result<ToolResult> {
 match call.name.as_str() {
 // ... existing tools
 "my_tool" => {
 let input = call.arguments["input"].as_str().unwrap();
 Ok(ToolResult::Success(format!("Processed: {}", input)))
 }
 _ => Err(anyhow!("Unknown tool: {}", call.name)),
 }
}
```

## Adding a Custom Provider

The `genai` crate handles providers. To add custom support:

1. **Set environment variable** for new provider
2. **Use provider-specific model names**

For custom API endpoints, modify `perspt-core/src/llm_provider.rs`.

## Adding TUI Components

1. **Create widget in perspt-tui:** `crates/perspt-tui/src/my_widget.rs`

```
use ratatui::{prelude::*, widgets::*};

pub struct MyWidget {
 data: String,
}

impl MyWidget {
 pub fn new(data: String) -> Self {
 Self { data }
 }

 pub fn render(&self, frame: &mut Frame, area: Rect) {
 let block = Block::default().title("My Widget").borders(Borders::ALL);
 let paragraph = Paragraph::new(self.data.clone()).block(block);
 frame.render_widget(paragraph, area);
 }
}
```

2. **Register in lib.rs:**

```
pub mod my_widget;
pub use my_widget::MyWidget;
```

## Adding Policy Rules

Extend the Starlark policy engine in `crates/perspt-policy/src/engine.rs`:

```
pub fn add_custom_rule(&mut self, pattern: &str, action: Action) {
 self.rules.push(Rule {
 pattern: pattern.to_string(),
 action,
 reason: None,
 });
}
```

## Testing Extensions

```
Test specific crate
cargo test -p perspt-agent

Run all tests
cargo test --all
```

(continues on next page)



(continued from previous page)

```
With coverage
cargo tarpaulin
```

## Documentation

Update docs when extending:

1. Update API docs in docs/perspt\_book/source/api/
2. Add usage examples to relevant tutorials
3. Rebuild: `cd docs/perspt_book && make html`

## See Also

- *Architecture* - Crate design
- *Testing* - Testing guide
- *API Reference* - API reference

### 1.12.4 Testing

Testing strategies for Perspt development.

## Running Tests

```
All tests
cargo test --all

Specific crate
cargo test -p perspt-agent

Specific test
cargo test test_name

With output
cargo test -- --nocapture
```

## Test Organization

Each crate has its own tests:

```
crates/
├── perspt-agent/
│ ├── src/
│ │ ├── orchestrator.rs
│ │ └── orchestrator_test.rs # Unit tests
│ └── tests/
│ └── integration_test.rs # Integration tests
```

## Unit Tests

Test individual functions in the same file:

```
// orchestrator.rs

#[cfg(test)]
mod tests {
 use super::*;

 #[test]
 fn test_energy_calculation() {
 let energy = Energy::compute(
 &[Diagnostic::error("test")],
 &TestResults::default(),
 1.0, 0.5, 2.0,
);
 assert!(energy.v_syn > 0.0);
 }

 #[tokio::test]
 async fn test_sheafification() {
 let plan = TaskPlan::from_prompt("Create file").await.unwrap();
 assert!(!plan.nodes.is_empty());
 }
}
```

## Integration Tests

Test crate interactions in tests/:

```
// tests/integration_test.rs

use perspt_agent::SRBNOrchestrator;
use perspt_core::GenAIProvider;
use std::sync::Arc;

#[tokio::test]
async fn test_full_workflow() {
 let provider = Arc::new(GenAIProvider::new().unwrap());
 let mut orchestrator = SRBNOrchestrator::new(
 provider,
 "./test_workspace".into(),
 Default::default(),
).await.unwrap();

 let result = orchestrator.execute("Create hello.py").await;
 assert!(result.is_ok());
}
```

## Mocking

Use mockall for mocking:

```

use mockall::mock;

mock! {
 pub LspClient {
 async fn get_diagnostics(&self, path: &Path) -> Result<Vec<Diagnostic>>;
 }
}

#[tokio::test]
async fn test_with_mock_lsp() {
 let mut mock_lsp = MockLspClient::new();
 mock_lsp
 .expect_get_diagnostics()
 .returning(|_| Ok(vec![]));

 // Use mock in test
}

```

## Coverage

```

Install tarpaulin
cargo install cargo-tarpaulin

Run with coverage
cargo tarpaulin --out Html

Open report
open tarpaulin-report.html

```

## Documentation Tests

Test code examples in docs:

```
cargo test --doc
```

## Benchmarks

Performance benchmarks in benches/:

```

// benches/energy_bench.rs

use criterion::{criterion_group, criterion_main, Criterion};
use perspt_agent::Energy;

fn energy_benchmark(c: &mut Criterion) {
 c.bench_function("energy_compute", |b| {
 b.iter(|| Energy::compute(&[], &Default::default(), 1.0, 0.5, 2.0))
 });
}

```

(continues on next page)

(continued from previous page)

```
criterion_group!(benches, energy_benchmark);
criterion_main!(benches);
```

```
cargo bench
```

## CI Testing

Tests run in GitHub Actions on every PR:

```
.github/workflows/test.yml
name: Tests
on: [push, pull_request]
jobs:
 test:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v4
 - uses: dtolnay/rust-toolchain@stable
 - run: cargo test --all
```

## See Also

- *Contributing* - How to contribute
- *Architecture* - Crate design

## 1.13 Changelog

All notable changes to Perspt will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

### 1.13.1 [0.4.6] - 2025-06-29

#### Added

- ☐ ☐ **Simple CLI Mode (PSP-000003)**: Brand new minimal command-line interface for direct Q&A without TUI overlay
  - `--simple-cli` flag enables Unix-style prompt interface perfect for scripting and automation
  - `--log-file <FILE>` option provides built-in session logging with timestamps
  - Real-time streaming responses in simple text format for immediate feedback
  - Clean exit handling with `Ctrl+D`, `exit` command, or `Ctrl+C` interrupt
  - Works seamlessly with all existing providers, models, and authentication methods
  - Perfect for accessibility needs, scripting workflows, and Unix philosophy adherents
- **Scripting and Automation Support**: Simple CLI mode enables powerful automation scenarios
  - Pipe questions directly: `echo "Question?" | perspt --simple-cli`
  - Batch processing with session logging for documentation and audit trails
  - Environment integration with aliases and shell scripts
  - Robust error handling that doesn't terminate sessions
- **Enhanced Accessibility**: Simple CLI mode provides better screen reader compatibility and simpler interaction model

## Technical Details

- New `cli.rs` module implementing async command-line loop with streaming support
- Integration with existing `GenAIProvider` for consistent behavior across interface modes
- Comprehensive error handling with graceful degradation for individual request failures
- Session logging format compatible with standard text processing tools

## Changed

- **Enhanced CLI Argument Support:** Added `--simple-cli` and `--log-file` arguments with proper validation
- **Updated Documentation:** Comprehensive updates to README and Perspt book covering simple CLI mode usage
- **Improved Help Text:** Clear descriptions of new simple CLI mode options and use cases

## Examples and Use Cases

```
Basic usage
perspt --simple-cli

With session logging
perspt --simple-cli --log-file research-session.txt

Scripting integration
echo "Explain quantum computing" | perspt --simple-cli

Environment setup
alias ai="perspt --simple-cli"
ai-log() { perspt --simple-cli --log-file "$1"; }
```

### 1.13.2 [0.4.2] - 2025-06-09

#### Added

- 🤖 **Zero-Config Automatic Provider Detection:** Perspt now automatically detects and configures available providers based on environment variables
  - Set any supported API key (`OPENAI_API_KEY`, `ANTHROPIC_API_KEY`, etc.) and simply run `perspt`
  - No configuration files or CLI arguments needed for basic usage
  - Intelligent priority-based selection: OpenAI → Anthropic → Gemini → Groq → Cohere → XAI → DeepSeek → Ollama
  - Automatic default model selection for each detected provider
  - Graceful fallback with helpful error messages when no providers are found
- **Enhanced Error Handling:** Clear, actionable error messages when no providers are configured
- **Comprehensive Provider Support:** All major LLM providers now supported for automatic detection
- **Local Model Auto-Detection:** Ollama automatically detected when running locally (no API key required)

#### Changed

- **Improved User Experience:** Launch Perspt instantly with just an API key - no config required
- **Better Documentation:** Updated getting-started guide and configuration documentation with zero-config examples
- **Streamlined Workflow:** Reduced friction for new users getting started

## Technical Details

- Added `detect_available_provider()` function in `config.rs` for environment-based provider detection
- Enhanced `load_config()` to use automatic detection when no explicit configuration is provided
- Comprehensive test coverage for all provider detection scenarios and edge cases

### 1.13.3 [0.4.1] - 2025-06-03

#### Added

- Enhanced documentation with Sphinx
- Comprehensive API reference
- Developer guide for contributors

#### Changed

- Improved error messages for better user experience
- Optimized memory usage for large conversations

#### Fixed

- Fixed terminal cleanup on panic
- Resolved configuration file parsing edge cases

### 1.13.4 [0.4.0] - 2025-05-29

#### Added

- **Multi-provider support:** OpenAI, Anthropic, Google, Groq, Cohere, XAI, DeepSeek, and Ollama
- **Dynamic model discovery:** Automatic detection of available models
- **Input queuing:** Type new messages while AI is responding
- **Markdown rendering:** Rich text formatting in terminal
- **Streaming responses:** Real-time display of AI responses
- **Comprehensive configuration:** JSON files and environment variables
- **Beautiful terminal UI:** Powered by Ratatui with modern design
- **Graceful error handling:** User-friendly error messages and recovery

#### Technical Highlights

- Built with Rust for maximum performance and safety
- Leverages *genai* crate for unified LLM access
- Async/await architecture with Tokio
- Comprehensive test suite with unit and integration tests
- Memory-safe with zero-copy operations where possible

#### Supported Providers

- **OpenAI:** GPT-4, GPT-4-turbo, GPT-4o series, GPT-3.5-turbo
- **Anthropic:** Claude 3 models (via *genai*)
- **Google:** Gemini models (via *genai*)
- **Groq:** Ultra-fast Llama inference
- **Cohere:** Command R/R+ models
- **XAI:** Grok models
- **DeepSeek:** Advanced reasoning models
- **Ollama:** Local model hosting

### Configuration Features

- Multiple configuration file locations
- Environment variable support
- Command-line argument overrides
- Provider-specific settings
- UI customization options

### User Interface Features

- Real-time chat interface
- Syntax highlighting for code blocks
- Scrollable message history
- Keyboard shortcuts for productivity
- Status indicators and progress feedback
- Responsive design that adapts to terminal size

## 1.13.5 [0.3.0] - 2025-05-15

### Added

- Multi-provider foundation with genai crate
- Configuration file validation
- Improved error categorization

### Changed

- Refactored provider architecture for extensibility
- Enhanced UI responsiveness
- Better handling of long responses

### Fixed

- Terminal state cleanup on unexpected exit
- Configuration merging precedence
- Memory leaks in streaming responses

## 1.13.6 [0.2.0] - 2025-05-01

### Added

- Streaming response support
- Basic configuration file support
- Terminal UI with Ratatui
- OpenAI provider implementation

### Changed

- Migrated from simple CLI to TUI interface
- Improved async architecture
- Better error handling patterns

### Fixed

- Terminal rendering issues
- API request timeout handling
- Configuration loading edge cases

## 1.13.7 [0.1.0] - 2025-04-15

### Added

- Initial release
- Basic OpenAI integration
- Simple command-line interface
- Environment variable configuration
- Basic chat functionality

### Features

- Support for GPT-3.5 and GPT-4 models
- API key authentication
- Simple text-based conversations
- Basic error handling

## 1.13.8 Migration Guides

### Upgrading from 0.3.x to 0.4.0

#### Configuration Changes:

The configuration format has been enhanced. Old configurations will continue to work, but consider updating:

```
// Old format (still supported)
{
 "api_key": "sk-...",
 "model": "gpt-4"
}

// New format (recommended)
{
 "api_key": "sk-...",
 "default_model": "gpt-4o-mini",
 "provider_type": "openai",
 "providers": {
 "openai": "https://api.openai.com/v1"
 }
}
```

#### Command Line Changes:

Some command-line flags have been updated:

```
Old
perspt --model gpt-4

New
perspt --model-name gpt-4
```



**API Changes:**

If you're using Perspt as a library, some function signatures have changed:

```
// Old
provider.send_request(message, model).await?;

// New
provider.send_chat_request(message, model, &config, &tx).await?;
```

**Upgrading from 0.2.x to 0.3.0****New Dependencies:**

Update your *Cargo.toml* if building from source:

```
[dependencies]
tokio = { version = "1.0", features = ["full"] }
... other dependencies updated
```

**Configuration Location:**

Configuration files now support multiple locations. Move your config file to:

- `~/.config/perspt/config.json` (Linux)
- `~/Library/Application Support/perspt/config.json` (macOS)
- `%APPDATA%/perspt/config.json` (Windows)

**1.13.9 Breaking Changes****Version 0.4.0**

- **Provider trait changes:** *LLMProvider* trait now requires *async fn* methods
- **Configuration structure:** Some configuration keys renamed for consistency
- **Error types:** Custom error types replace generic error handling
- **Streaming interface:** Response handling now uses channels instead of callbacks

**Version 0.3.0**

- **Async runtime:** Switched to full async architecture
- **UI framework:** Migrated from custom rendering to Ratatui
- **Configuration format:** Enhanced JSON schema with validation

**Version 0.2.0**

- **Interface change:** Moved from CLI to TUI
- **Provider abstraction:** Introduced provider trait system
- **Async support:** Added Tokio async runtime

**1.13.10 Deprecation Notices**

The following features are deprecated and will be removed in future versions:

### Version 0.5.0 (Upcoming)

- **Legacy configuration keys:** Old configuration format support will be removed
- **Synchronous API:** All provider methods must be async
- **Direct model specification:** Use provider + model pattern instead

### Version 0.6.0 (Planned)

- **Environment variable precedence:** Will change to match command-line precedence
- **Default provider:** Will change from OpenAI to provider-agnostic selection

## 1.13.11 Known Issues

### Current Version (0.4.0)

- **Windows terminal compatibility:** Some Unicode characters may not display correctly on older Windows terminals
- **Large conversation history:** Memory usage increases with very long conversations (>1000 messages)
- **Network interruption:** Streaming responses may be interrupted during network issues
- **Ollama connectivity:** Local models may require manual service restart after system reboot

Workarounds:

```
For Windows terminal issues
Use Windows Terminal or enable UTF-8 support

For memory issues with large histories
perspt --max-history 500

For network issues
perspt --timeout 60 --max-retries 5
```

## 1.13.12 Planned Features

### Version 0.5.0 (Next Release)

- **Local model support:** Integration with Ollama and other local LLM servers
- **Plugin system:** Support for custom providers and UI extensions
- **Conversation persistence:** Save and restore chat sessions
- **Multi-conversation support:** Multiple chat tabs in single session
- **Enhanced markdown:** Tables, math equations, and diagrams
- **Voice input:** Speech-to-text support for hands-free operation

### Version 0.6.0 (Future)

- **Collaborative features:** Share conversations and collaborate with others
- **IDE integration:** VS Code extension and other editor plugins
- **Mobile companion:** Mobile app for conversation sync
- **Advanced AI features:** Function calling, tool use, and agent capabilities
- **Performance analytics:** Response time tracking and optimization suggestions

### Version 1.0.0 (Stable Release)

- **API stability guarantee:** Stable public API with semantic versioning
- **Enterprise features:** SSO, audit logging, and compliance features
- **Advanced customization:** Themes, layouts, and workflow customization
- **Comprehensive integrations:** GitHub, Slack, Discord, and more
- **Professional support:** Documentation, training, and enterprise support

### 1.13.13 Contributing

We welcome contributions! Please see our [Contributing](#) for guidelines.

**Types of contributions:** - Bug reports and feature requests - Code contributions and optimizations - Documentation improvements - Testing and quality assurance - Community support and advocacy

#### How to contribute:

1. Check existing issues and discussions
2. Fork the repository
3. Create a feature branch
4. Make your changes with tests
5. Submit a pull request

### 1.13.14 Support

- **GitHub Issues:** [Bug Reports](#)
- **Discussions:** [Community Chat](#)
- **Documentation:** This guide and API reference
- **Email:** [support@perspt.dev](mailto:support@perspt.dev) (for enterprise inquiries)

### 1.13.15 License

Perspt is released under the LGPL v3 License. See [License](#) for details.

### 1.13.16 Acknowledgments

Special thanks to:

- The Rust community for excellent tooling and libraries
- Ratatui developers for the amazing TUI framework
- genai crate maintainers for unified LLM access
- All contributors and users who help improve Perspt

#### See also

- [Installation Guide](#) - How to install or upgrade Perspt
- [Getting Started](#) - Quick start guide for new users
- [Contributing](#) - How to contribute to the project

## 1.14 License

Perspt is released under the GNU Lesser General Public License v3.0 (LGPL v3).

### 1.14.1 LGPL v3 License

Copyright (c) 2025 Ronak Rathoer, Vikrant Rathore

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this program. If not, see <<https://www.gnu.org/licenses/>>.

### 1.14.2 What This Means

The LGPL v3 is a copyleft license that provides strong protection for software freedom while allowing linking with proprietary software. Here's what it means in practical terms:

□ **What you CAN do:**

- **Use** Perspt for any purpose, including commercial projects
- **Modify** the source code to fit your needs
- **Distribute** copies of Perspt
- **Link** Perspt as a library in proprietary software
- **Combine** Perspt with software under different licenses
- **Create** derivative works based on Perspt

□ **What you MUST do:**

- **Provide source code** for any modifications to Perspt itself
- **Include** the LGPL v3 license text with distributions
- **Preserve** copyright notices and license information
- **Allow users** to replace the Perspt library with modified versions
- **Make modified source** available under LGPL v3 terms

⊘ **What we DON'T provide:**

- **Warranty** - The software is provided "as is"
- **Liability coverage** - We're not responsible for any damages
- **Support guarantees** - While we strive to help, support is provided on a best-effort basis

### 1.14.3 Third-Party Licenses

Perspt depends on several open source libraries, each with their own licenses:

#### Core Dependencies

| Crate            | License        | Description                     |
|------------------|----------------|---------------------------------|
| <b>tokio</b>     | MIT            | Async runtime for Rust          |
| <b>ratatui</b>   | MIT            | Terminal user interface library |
| <b>serde</b>     | MIT/Apache-2.0 | Serialization framework         |
| <b>clap</b>      | MIT/Apache-2.0 | Command line argument parser    |
| <b>anyhow</b>    | MIT/Apache-2.0 | Error handling library          |
| <b>thiserror</b> | MIT/Apache-2.0 | Error derive macros             |

## LLM Integration

| Crate          | License        | Description                    |
|----------------|----------------|--------------------------------|
| <b>genai</b>   | MIT/Apache-2.0 | Unified LLM provider interface |
| <b>request</b> | MIT/Apache-2.0 | HTTP client library            |

## Terminal and UI

| Crate                | License        | Description                         |
|----------------------|----------------|-------------------------------------|
| <b>crossterm</b>     | MIT            | Cross-platform terminal library     |
| <b>unicode-width</b> | MIT/Apache-2.0 | Unicode character width calculation |
| <b>textwrap</b>      | MIT            | Text wrapping and formatting        |

## Development Dependencies

| Crate            | License        | Description               |
|------------------|----------------|---------------------------|
| <b>criterion</b> | MIT/Apache-2.0 | Benchmarking library      |
| <b>mockall</b>   | MIT/Apache-2.0 | Mock object library       |
| <b>tempfile</b>  | MIT/Apache-2.0 | Temporary file management |

### 1.14.4 License Compatibility

The LGPL v3 is compatible with most other open source licenses:

**Compatible Licenses:** - Apache License 2.0 - BSD licenses (2-clause, 3-clause) - ISC License - MIT License - Public Domain (CC0) - GPL v3+ (can be upgraded to GPL)

**Special Considerations:** - GPL v2: Not directly compatible due to version differences - Proprietary licenses: Can link with LGPL libraries but must allow library replacement - Copyleft licenses: LGPL provides weaker copyleft than GPL

### 1.14.5 Commercial Use

Perspt can be freely used in commercial projects:

#### □ Allowed Commercial Uses:

- **Internal tools** - Use Perspt as part of your development workflow
- **Linked libraries** - Link Perspt as a library in commercial software
- **Service offerings** - Provide Perspt as part of consulting or hosting services
- **Modified library versions** - Create modified versions for internal use
- **Enterprise solutions** - Build enterprise tools that use Perspt

#### 🔒 Requirements for Commercial Use:

1. **Include LGPL license text** in your distribution
2. **Maintain copyright notices** from the original code
3. **Provide source code** for any modifications to Perspt itself
4. **Allow library replacement** - users must be able to replace the Perspt library
5. **No trademark usage** without permission (see below)

No additional fees, registrations, or permissions are required.

### 1.14.6 Trademark Policy

While the source code is LGPL v3 licensed, trademarks are handled separately:

**“Perspt” Name and Logo:** - The name “Perspt” and any associated logos are trademarks - You may use the name in accurately describing the software - Commercial use of the name/logo as your own brand requires permission - Modified versions should use different names to avoid confusion

**Acceptable Uses:** - “Built with Perspt” - “Based on Perspt” - “Powered by Perspt” - “Fork of Perspt”

**Requires Permission:** - Using “Perspt” as your product name - Using Perspt logos in your branding - Implying official endorsement

### 1.14.7 Contributing and License

By contributing to Perspt, you agree that:

1. **Your contributions** will be licensed under the same LGPL v3 License
2. **You have the right** to license your contributions under LGPL v3
3. **You understand** that your contributions may be used commercially
4. **You retain copyright** to your contributions while granting broad usage rights

#### Contributor License Agreement (CLA)

For substantial contributions, we may request a Contributor License Agreement to:

- Ensure you have the right to contribute the code
- Provide legal protection for the project and users
- Allow for potential future license changes if needed
- Clarify the rights and responsibilities of contributors

### 1.14.8 License FAQ

**Q: Can I use Perspt in my proprietary software?** A: Yes, LGPL v3 allows linking with proprietary software. You must provide the library source and allow replacement.

**Q: Can I modify Perspt and sell the modified version?** A: Yes, but you must provide the source code for your modifications under LGPL v3.

**Q: Do I need to open source my modifications?** A: Yes, any modifications to Perspt itself must be made available under LGPL v3.

**Q: Can I remove the copyright notices?** A: No, you must preserve the copyright notices and license information in all copies.

**Q: What if I only use parts of the code?** A: The LGPL v3 license still applies to any substantial portions you use.

**Q: Can I change the license of my derivative work?** A: You can license your own code separately, but Perspt parts must remain LGPL v3.

**Q: Do I need to attribute Perspt in my application?** A: Yes, you must include the LGPL v3 license and copyright notices.

### 1.14.9 Getting Legal Advice

This page provides general information about the LGPL v3 License and is not legal advice. For specific legal questions:

- **Consult** with a qualified attorney
- **Review** the full license text carefully
- **Consider** your specific use case and jurisdiction
- **Seek** professional legal counsel for commercial decisions

### 1.14.10 Reporting License Issues

If you believe there's a license violation or have questions about licensing:

- **Email:** [legal@perspt.dev](mailto:legal@perspt.dev)
- **GitHub Issues:** [License Questions](#)
- **Include** specific details about the concern or question

We take licensing seriously and will investigate all reports promptly.

#### ➡ See also

- [Acknowledgments](#) - Credits and thanks to contributors
- [Contributing](#) - How to contribute to the project
- [GNU Project](#) - Official LGPL v3 License text

## 1.15 Acknowledgments

Perspt is built on the shoulders of giants. We extend our gratitude to the many open-source projects, libraries, and communities that made this project possible.

### 1.15.1 Core Dependencies

#### AI and LLM Integration

##### genai

The foundation of Perspt's multi-provider support. This exceptional crate provides unified interfaces to multiple AI providers and automatically stays up-to-date with new models and capabilities.

- **Project:** [genai](#)
- **License:** MIT/Apache 2.0
- **Impact:** Enables seamless integration with OpenAI, Anthropic, Google, Groq, Cohere, XAI, DeepSeek, and Ollama providers

##### serde & serde\_json

Rust's premier serialization framework, powering Perspt's configuration management and API communication.

- **Project:** [serde](#)
- **License:** MIT/Apache 2.0
- **Impact:** JSON configuration parsing, API request/response handling

#### User Interface and Terminal

##### ratatui

The modern, feature-rich TUI framework that powers Perspt's interactive terminal interface.

- **Project:** [ratatui](#)
- **License:** MIT
- **Impact:** Rich terminal UI, markdown rendering, scrollable chat interface

##### crossterm

Cross-platform terminal manipulation library enabling consistent behavior across operating systems.

- **Project:** [crossterm](#)
- **License:** MIT
- **Impact:** Keyboard input handling, terminal control, cross-platform compatibility

## Async Runtime and Concurrency

### tokio

The asynchronous runtime that enables Perspt’s responsive, non-blocking architecture.

- **Project:** [tokio](#)
- **License:** MIT
- **Impact:** Async/await support, concurrent LLM requests, responsive UI

## Error Handling and Utilities

### anyhow

Elegant error handling that makes Perspt’s error messages helpful and actionable.

- **Project:** [anyhow](#)
- **License:** MIT/Apache 2.0
- **Impact:** Comprehensive error context, user-friendly error messages

### clap

Command-line argument parsing that makes Perspt easy to use and configure.

- **Project:** [clap](#)
- **License:** MIT/Apache 2.0
- **Impact:** CLI interface, help generation, argument validation

## 1.15.2 Documentation Tools

### Sphinx

The documentation generator that created this beautiful book-style documentation.

- **Project:** [Sphinx](#)
- **License:** BSD
- **Impact:** Professional documentation, PDF generation, cross-references

### Furo Theme

The modern, accessible Sphinx theme that makes this documentation a pleasure to read.

- **Project:** [Furo](#)
- **License:** MIT
- **Impact:** Beautiful documentation design, responsive layout, accessibility

## 1.15.3 Development Tools

### Rust Language

The systems programming language that makes Perspt fast, safe, and reliable.

- **Project:** [Rust](#)
- **License:** MIT/Apache 2.0
- **Impact:** Memory safety, performance, excellent tooling ecosystem

### cargo

Rust’s package manager and build system that makes development smooth and dependency management effortless.

- **Project:** Part of Rust toolchain
- **License:** MIT/Apache 2.0
- **Impact:** Dependency management, build automation, testing framework

## 1.15.4 Community and Inspiration

### AI Provider Communities

#### OpenAI

For creating GPT models and establishing many of the patterns that define modern AI interaction.

#### Anthropic

For Claude models and their pioneering work in AI safety and helpful, harmless, and honest AI systems.



### Google

For Gemini models and their contributions to accessible AI technology.

### Groq

For ultra-fast inference infrastructure and democratizing AI speed.

### Cohere

For enterprise-grade language models and excellent developer tools.

### XAI

For Grok models and advancing conversational AI capabilities.

### DeepSeek

For their contributions to the open-source AI ecosystem.

### Ollama

For making local AI model hosting accessible and user-friendly.

## Open Source Ecosystem

### GitHub

For providing the platform that enables collaborative development and open-source sharing.

### crates.io

Rust's package registry that makes sharing and discovering Rust libraries effortless.

### docs.rs

For automatically generating and hosting documentation for Rust crates.

## Terminal and CLI Inspiration

The terminal and CLI interface draws inspiration from many excellent tools:

- **htop** - For showing how terminal UIs can be both beautiful and functional
- **tmux** - For terminal multiplexing concepts and keyboard navigation patterns
- **vim/neovim** - For modal editing concepts and efficient keyboard shortcuts
- **fzf** - For demonstrating responsive, interactive terminal interfaces

## Rust Community Projects

Many patterns and approaches in Perspt were learned from studying excellent Rust projects:

- **ripgrep** - For performance optimization and user experience design
- **bat** - For beautiful terminal output and syntax highlighting
- **exa/eza** - For modern CLI design and colored output
- **gitui** - For TUI application architecture and event handling

## 1.15.5 Testing and Quality Assurance

### Users and Beta Testers

The early adopters and users who provided feedback, reported bugs, and suggested improvements.

### Security Researchers

For responsible disclosure of security issues and helping make Perspt more secure.

### Documentation Reviewers

For helping improve the clarity and completeness of this documentation.

## 1.15.6 Special Thanks

### AI Safety Research Community

For ongoing work to make AI systems more reliable, interpretable, and aligned with human values.

### Open Source Contributors

To everyone who contributes to open-source projects, from major features to documentation fixes.

### **Rust Community**

For creating and maintaining an inclusive, helpful community that makes Rust development a joy.

### **Terminal Enthusiasts**

For keeping the art of terminal-based applications alive and pushing the boundaries of what's possible in text-based interfaces.

## **1.15.7 Contributing Back**

Perspt aims to be a good citizen of the open-source ecosystem. We contribute back by:

### **Open Source Release**

Perspt itself is released under the LGPL v3 license, allowing anyone to use, modify, and distribute it.

### **Documentation Standards**

This comprehensive documentation serves as an example of thorough project documentation.

### **Best Practices Sharing**

Through blog posts, talks, and code examples, we share what we've learned building Perspt.

### **Upstream Contributions**

When we find bugs or missing features in dependencies, we contribute fixes and improvements back to those projects.

## **1.15.8 License Information**

Perspt is licensed under the LGPL v3 License. For complete license information, see [License](#).

All dependencies are used in accordance with their respective licenses. We are grateful to all the authors and maintainers who choose to share their work under permissive open-source licenses.

## **1.15.9 Get Involved**

Want to contribute to Perspt or the broader ecosystem?

### **Report Issues**

Help improve Perspt by reporting bugs, suggesting features, or improving documentation.

### **Contribute Code**

See our [Contributing](#) guide for how to contribute code improvements.

### **Share Knowledge**

Write blog posts, create tutorials, or give talks about your experience with Perspt.

### **Support Dependencies**

Consider contributing to the open-source projects that Perspt depends on.

### **Spread the Word**

Help others discover Perspt and the amazing ecosystem of Rust and AI tools.

—

*Thank you to everyone who makes open-source software development possible. Your contributions, large and small, make projects like Perspt possible.*

## Chapter 2

### Quick Links

- [GitHub Repository](#)
- [Crates.io](#)
- [PSP Process](#)
- [Issue Tracker](#)



## Chapter 3

# Indices

- `genindex`
- `modindex`
- `search`



# Index

## B

built-in function  
    run\_agent\_tui(), 131

## R

run\_agent\_tui()  
    built-in function, 131